Derivation of access request arrival curves for dedicated superblock sequences

TIK Report No. 347

Georgia Giannopoulou, Nikolay Stoimenov, Kai Lampka,
Andreas Schranzhofer, Lothar Thiele
Computer Engineering and Networks Laboratory, ETH Zurich
lastname@tik.ee.ethz.ch

Abstract

This document is intended to complement the work presented in [4]. In particular, it provides a more elaborate documentation of the method used for the derivation of arrival curves which bound the access request streams of periodically executed dedicated superblocks (Sec. 4.2.1, [4]). Additionally, it presents several abstractions that have been applied to the timed automata-based modelling of a resource-sharing multicore system in order to alleviate the complexity of model checking (extension to Sec. 4.2.3, [4]).

1 Introduction

The work presented in [4] addresses the problem of analysing the worst-case response time (WCRT) of tasks which are executed in parallel on a multicore system and can access shared resources, such as cache memories or interconnection buses. Access requests to those resources are synchronous (blocking), namely execution of a task may need to stall until a resource is available (not accessed by any other task). This interference of the tasks, which is imposed by the utilization of shared resources, causes waiting times that increase their WCRT and therefore need to be bounded tightly. The latter, however, is not trivial since it depends on the arbitration policy of the resources, which may be of any complexity.

The suggested WCRT analysis methodology, which uses timed automata to model the resource contention scenarios and model checking techniques to compute the tasks' WCRT, is based on the following system assumptions:

- A system consists of several processing cores $p_j \in P$, which execute independent tasks, but can access a common resource. The task partitioning (mapping of tasks onto cores) is predefined and no task migration is allowed during runtime.

- The tasks that are mapped on a particular core p_j are executed periodically according to a static schedule, which is repeated with period W_j (processing cycle). Furthermore, the tasks are specified by a sequence of *superblocks* S_j, which are non-preemptable execution units with known lower and upper bounds on their computation time and number of required resource accesses. The superblocks in S_j are executed sequentially, i. e., in their relative order, which is defined by the static schedule of p_j, each superblock $s_{i+1,j}$ is triggered upon completion of its predecessor $s_{i,j}$ ($i \in [1, |S_j| - 1]$).
- Each superblock in S_j is defined according to the dedicated model [8]. That is, superblocks are separated into three phases, known as *acquisition* (A), *execution* (E), and *replication* (R): A superblock reads the required data during its acquisition phase and writes back the modified/new data in the replication phase, after computations in the execution phase have been completed. The acquisition and replication phases, during which resource accesses are performed sequentially, are characterized by their minimum and maximum number of access requests. On the other hand, the execution phase, during which no resource accesses are enabled, is characterized by its minimum and maximum computation time.
- A shared resource can be accessed by at most one core (task) at a time. Accesses are non preemptable and of constant duration, C. Their enabling is organized by a resource arbiter, which may be of arbitrary complexity, implementing for example a first come first served (FCFS), round-robin (RR), fixed priority (FP), time division multiple access (TDMA) or even a more complex arbitration scheme, such as the industrial FlexRay bus protocol [1].
- The multicore hardware platform is such that computation time and communication time can be decoupled (such as in the fully timing compositional architecture proposed in [11]). Namely, the hardware platform is assumed to have no timing anomalies.

For a system complying with the above assumptions, the parallel task execution and resource sharing can be precisely modelled using the formalism of timed automata [2]. Such a model of the system can, subsequently, be analyzed with a timed model checker, as for instance Uppaal [3]. The model checker explores exhaustively all feasible interleavings of the tasks (superblocks) that can occur in runtime and therefore, all feasible interference scenarios over the shared resources. As a result, the delays that each task suffers while attempting to access the resources can be accurately bounded and so can also its WCRT.

The benefit of accuracy for this WCRT analysis approach is of particular importance, since most existing analytic approaches that address the problem of timing analysis in multicore resource-sharing systems are based on several abstractions of the resource arbitration mechanism (esp. in case of

event-driven arbitration), which lead to very pessimistic WCRT estimations, see e. g., [6, 7, 9].

On the other hand, the suggested model checking-based WCRT analysis approach suffers from scalability issues, given that the complexity of model checking increases exponentially with the size of the system (e. g., number of cores on which tasks are executed in parallel). To cope with this challenge, [4] presents an extension to the initial method, which uses jointly timed automata and the real-time calculus [10] to model the multicore resource-sharing system. In particular, task execution and resource accessing on several cores are modeled through arrival curves [5]. An arrival curve bounds the maximum number of access requests that the tasks of a core can issue in any time interval, thus providing a means to represent (abstract) the accessing pattern on each core. This information can be then expressed with timed automata. What is important is that the number of timed automata required to model this information is constant and not dependent on the number of tasks (superblocks) that are executed on each core like in the initial method. This results in a model of the system with significantly less components (timed automata), which helps alleviate the complexity of model checking.

In the following, Sec. 2 presents the method used to derive an arrival curve representing the resource accessing pattern on a core, when the tasks executing on it are specified as sequences of dedicated superblocks. This method was introduced in [6] for the case of general superblock sequences (superblocks in which computation and resource accesses can happen any time) and was later refined in [4] for dedicated superblock sequences. Sec. 2 elaborates on the method presented in [4], illustrating it with examples.

Additionally, Sec. 3 presents several abstractions that have been applied to the timed-automata based model of the studied systems, on top of the major arrival curve abstraction. All of them are aimed to reduce the size of the model and hence, to alleviate the complexity of model checking and boost analysis scalability.

2 From superblocks to access request arrival curves

A set of superblocks S_j executing on processing core p_j, with processing cycle W_j, accesses a shared resource according to a pattern. This pattern is specified by the minimum and maximum number of access requests and the minimum and maximum computation time of each superblock in S_j. In this section, we show how to represent such an access pattern as an arrival curve, the latter providing an upper bound on the number of access requests that are issued by the corresponding core in any interval of time.

The arrival curve for a core p_j is derived assuming that no interference occurs on the resource. In other words, the superblock set of p_j is analyzed

P	set of processing cores	$F_{m,d}$	ordered subset of phases from $f_{m,j}$ to $f_{m+d,j}$
p_j	processing core j, $p_j \in P$	$\gamma^{y}_{m,d,k}$	access requests in k-th time window of $F_{m,d}$
W_j	processing cycle length on p_j	$\Delta^{x,y}_{m,d,k}$	length of k-th time window of $F_{m,d}$
S_j	superblock set mapped on p_j	g	min. gap between two consecutive instances of S_j
$s_{i,j}$	superblock i on core p_j, $s_{i,j} \in S_j$	$t^{x,y}_{m,d,k}$	tuples for k-th time window of $F_{m,d}$
C	resource access latency	$x \in${min,max}	min/max computation time of phases in a time window
$f_{i',j}$	superblock phase i' on p_j, $i' \in [1, 3 \cdot \lvert S_j \rvert]$	$y \in${min,max}	min/max access requests of phases in a time window
$\mu^{[min\lvert max]}_{i',j}$	min/max accesses requests in phase $f_{i',j}$	$\delta(t), \nu(t)$	window length & access requests of tuple t
$ex^{[min\lvert max]}_{i',j}$	min/max computation time in phase $f_{i',j}$	w_j	worst-case (min) period of access request stream on p_j

Table 1: Symbols for the system model and arrival curve derivation

in isolation, as if it had exclusive access to the resource. The arrival curve construction involves then the following steps:

1. *Computation of all ordered subsets of superblock phases within two consecutive processing cycles*: Based on the definition of the set S_j, which is an ordered set since the superblocks are statically scheduled on p_j, we compute all possible ordered subsets of phases within two processing cycles. Each phase subset yields multiple time windows during which new access requests can be issued.
2. *Computation of the time windows and the maximum number of access requests that can be issued during them for all phase subsets*: For each time window within a phase subset, this information is expressed as a tuple of the form (γ, Δ), when γ is the maximum number of requests that can occur in the time window of length Δ. Several time windows might have equal lengths but different number of access requests. Eventually, among them, only the window(s) with the highest number of access requests will contribute to the arrival curve.
3. *Construction of the arrival curve of core p_j*: An initial arrival curve is derived based on the previously computed tuples. Eventually, the arrival curve representation of p_j's access pattern will be a periodic repetition of this initial curve.

The three steps are discussed in more detail in the following subsections. For a summary of the variables that are involved in the computations the reader is referred to Table 1.

2.1 Computing sequences of superblock phases

Initially, we translate the superblock set S_j, of which the access pattern we want to represent, into the sequence of its constituent phases. Given the dedicated access model of the superblocks of S_j, that means that each superblock $s_{i,j} \in S_j$ can be translated into a sequence of three phases, which are denoted in the following as $f_{i'+1,j}$ (acquisition), $f_{i'+2,j}$ (execution) and $f_{i'+3,j}$ (replication), where $i' = 3 \cdot (i-1)$. Each phase $f_{i',j}$ is characterized by a minimum/maximum number of access requests and a minimum/maximum computation time, which are denoted in the following as $\mu_{i',j}^{[min|max]}$ and $ex_{i',j}^{[min|max]}$ respectively. Consider, for instance, the superblock s_1[1] of Fig. 1, which is mapped to its three constituent phases, f_1 with $[\mu_1^{min}, \mu_1^{max}] = [4, 6]$, f_2 with $[ex_2^{min}, ex_2^{max}] = [45, 60]\mu$s, and finally f_3 with $[\mu_3^{min}, \mu_3^{max}] = [1, 4]$[2]. Similarly, the superblock set S_j can be mapped to an ordered sequence of phases, i. e., $S_j = \{f_{1,j}, ..., f_{3 \cdot |S_j|,j}\}$.

The goal of the first step is to specify all ordered subsets of phases that can be executed within two consecutive processing cycles on p_j, e. g., within the time interval $[0, 2W_j]$. Note that we consider two instances of S_j in order to account for the transition phase (slack time) between the two processing cycles. To specify the relevant subsets, we first translate the superblock sequence $\{Sj, Sj\} = \{s_{1,j}, ..., s_{|Sj|,j}, s_{1,j}, ..., s_{|Sj|,j}\}$ into the phase sequence $\{f_{1,j}..., f_{3 \cdot |Sj|,j}, f_{1,j}, ..., f_{3 \cdot |Sj|,j}\}$. The latter corresponds to a set of $\frac{3 \cdot |S_j| (9 \cdot |S_j| + 1)}{2}$ unique ordered subsets of phases. Each ordered subset, denoted as $F_{m,d}$, is described by the index m of the first phase it contains and the distance d to the last phase, respectively, such that:

$$F_{m,d} = \{f_{m,j}, ..., f_{m+d,j}\}, \forall d \in [0...6 \cdot |S_j| - 1], \forall m \in [1...3 \cdot |S_j|] \quad (1)$$

Note that (a) we consider only those subsets with $m + d \leq 6 \cdot |S_j|$ and (b) that if $m + d > 3 \cdot |S_j|$, then phase $f_{m+d,j}$ is equivalent to $f_{(m+d)\%|S_j|,j}$ of the second processing cycle[3]. One may think of the phase subsets as sliding windows, including up to d successive superblock phases within two processing cycles. For example, for the two instances of the superblock set S in Fig. 1, which consists of just one superblock, s_1, the (15) possible ordered subsets are $F_{1,1}, F_{2,2}, F_{3,3}$ (each phase in isolation), $F_{1,2}, F_{2,3}, F_{3,4}$ (every subset of 2 successive phases), $F_{1,3}, F_{2,4}, F_{3,5}$ (3 successive phases), $F_{1,4}, F_{2,5}, F_{3,6}$ (4 successive phases), $F_{1,5}, F_{2,6}$ (5 successive phases), and finally $F_{1,6}$ (6 successive phases).

[1] The index of the processing core has been omitted for clarity of presentation.

[2] Note that due to the dedicated access model, $ex_1^{min} = ex_1^{max} = ex_3^{min} = ex_3^{max} = 0$ and $\mu_2^{min} = \mu_2^{max} = 0$.

[3] E.g., phase subset $F_{3,5}$ in Fig. 1 covers phases $f_{3,j}$ of the first and $f_{1,j}$, $f_{2,j}$ of the second processing cycle.

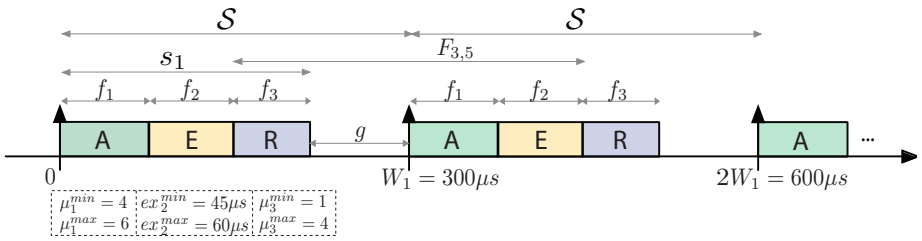

Figure 1: Two consecutive executions of superblock set S

2.2 Computing time windows and number of access requests within them

Every phase subset $F_{m,d}$ can be associated with several time windows with different number of access requests. As shown in the following, we can compute these time windows such that they are as short as possible while they contain as many access requests as possible. This way the windows represent the worst-case interference that the phase subset can cause to any other task attempting to access the shared resource within their duration.

Each time window is defined by a tuple (γ, Δ), where γ is the maximum amount of requests that can occur in its duration Δ. Let us consider the two instances of S in Fig. 1 and particularly, the phase subset $F_{1,1} = \{f_1\}$. It is easy to notice that the shortest time window that includes the first access request has a length of 0 (request is issued immediately upon start of f_1), which yields the tuple (1,0). Similarly, the second access request can be issued at earliest C time units after issuing the first one, namely immediately after the first is served, yielding the tuple (2, C). In the same way, we can obtain also the tuples (3, $2C$), (4, $3C$), and also (5, $4C$) and (6, $5C$) since $\mu_1 \in [4, 6]$. Next, we consider the phase subset $F_{1,2} = \{f_1, f_2\}$. For this subset, since f_2 is an execution phase, there are no time windows with newly issued access requests that have not been considered already. On the contrary, the phase subset $F_{1,3} = \{f_1, f_2, f_3\}$ results in new time windows. For instance, if we take $\mu_1 = \mu_1^{min} = 4$, a 5th access request can be issued in time windows with a length between $(4C + ex_2^{min})$ and $(4C + ex_2^{max})$, yielding e. g., the tuples (5, $4C + ex_2^{min}$) and (5, $4C + ex_2^{max}$). We can notice that these tuples correspond to longer time windows that the previously computed tuple (5, $4C$), so they will not contribute to the construction of the arrival curve, as shown in Sec. 2.3. A new tuple can be also computed if we consider the shortest time window that includes 7 access requests. For all possible values of $\mu_1 \in [4, 6]$, this time window has at least a length of $(6C + ex_2^{min})$, yielding the tuple (7, $6C + ex_2^{min}$), and so forth.

Generally, each phase subset $F_{m,d}$ results in up to $4 \cdot \mu_{m+d,j}^{max}$ new time windows on condition that its last phase, $f_{m+d,j}$, is an access (acquisition of

replication) phase. Otherwise, $F_{m,d}$ does not contribute any time windows that have not been considered already. In the first case, we seek new windows in $F_{m,d}$ which differ from each other by at least 1 access request. Such windows can be specified if we consider the accesses of the last phase $f_{m+d,j}$. Actually, each newly issued access request of $f_{m+d,j}$ can occur in up to 4 different time windows, if we consider the minimum and maximum access requests for each access phase and the minimum and maximum computation time for each execution phase in $F_{m,d}$. All these parameter combinations, which need to be examined to guarantee that the worst-case behavior is accounted for, lead to a total of $4 \cdot \mu^{max}_{m+d,j}$ windows per phase subset.

The time lengths $\Delta_{m,d,k}$ and the respective access requests $\gamma_{m,d,k}$ of the windows of $F_{m,d}$ are computed as in Eq. 2 and 3, $\forall k \in [1, \mu^{max}_{m+d,j}]$:

$$\Delta^{x,y}_{m,d,k} = \sum_{i=m+1}^{m+d-1} ex^{x}_{i,j} + \left(\sum_{i=m}^{m+d-1} \mu^{y}_{i,j} + k - 1 \right) \cdot C \tag{2}$$

$$\gamma^{y}_{m,d,k} = \sum_{i=m}^{m+d-1} \mu^{y}_{i,j} + k \tag{3}$$

Note that the superscripts $x = max$ or $x = min$ denote maximum or minimum computation time, and $y = max$ or $y = min$ maximum or minimum number or access requests for each phase, respectively.

Moreover, it is interesting to note that computing the time windows for subsets $F_{m,d}$, whose phase sequence spans over a processing cycle, needs to consider the gap (slack time) g between the last phase of S_j and the period W_j of the processing cycle. Example $F_{3,5}$ in Fig. 1 illustrates such a case. To represent the worst-case access behavior, we need to minimize the gap and deductively, the length of the time windows that include it. In fact, in the actual multicore system, where the superblocks of S_j will compete against superblocks of other cores for access to the shared resource, it is possible that the incurred delays will cause the execution of S_j to be extended up to the end of the processing cycle, as shown in Fig. 2. Therefore, unless some information (bounds) on these delays is provided, we need to consider $g = 0$ as the minimum feasible gap between two successive executions of S_j, in order to derive a safe arrival curve.

In conclusion, each phase subset $F_{m,d}$ is characterized by a set of tuples $t^{x,y}_{m,d,k}$(Eq. 4), i. e., a set of time windows and the respective number of access requests that can be issued by p_j within them. The tuples are defined for each $k \in [1, \mu^{max}_{m+d,j}]$ based on Eq. 2, 3 and the minimum feasible gap g, as follows:

$$t^{x,y}_{m,d,k} = (\gamma^{y}_{m,d,k}, \Delta^{x,y}_{m,d,k} + g) \tag{4}$$

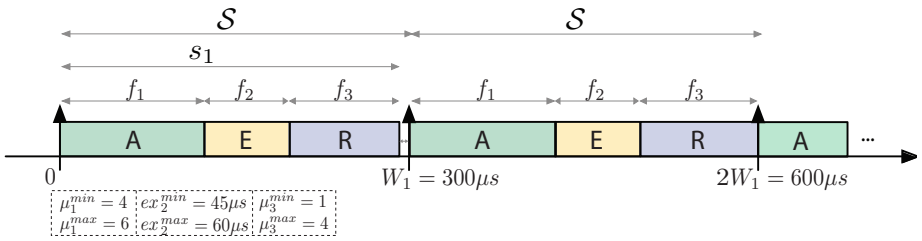

Figure 2: Two consecutive executions of superblock set S (worst-case): The WCRT of S is equal to W_j due to interference over shared resources.

2.3 Deriving the arrival curve

One can trivially notice that the worst-case access pattern of core p_j can occur when all superblocks in S_j emit the maximum number of access requests, $\mu_j^{max,tot} = \sum_{i=1}^{3\cdot|S_j|} \mu_{i,j}^{max}$, at every execution. This leads to a stream of $\mu_j^{max,tot}$ access requests which occurs with a minimum inter-arrival time that is given by Eq. 5 if we consider the minimum execution times of all superblocks and the minimum gap between any two successive execution instances of S_j:

$$w_j = \mu_j^{max,tot} \cdot C + \sum_{i=1}^{3\cdot|S_j|} ex_{i,j}^{min} + g \tag{5}$$

To derive a safe arrival curve that bounds all possible access request streams of p_j, one has to take into account the worst-case arrival pattern of them. That occurs obviously when the stream of $\mu_j^{max,tot}$ requests arrives *periodically* with a period equal to w_j (minimum inter-arrival time, Eq. 5)).

Based on the above observation, the initial part of the arrival curve can be constructed by retrieving the maximum number of access requests for every time interval $\Delta = \{0 \dots w_j\}$ from the computed tuples. In particular, if function $\delta(t)$ returns the length of the time window and $\nu(t)$ the number of access requests for each tuple t, then the upper arrival curve $\widetilde{\alpha}_j$, for $\Delta \in [0, w_j]$, can be obtained as:

$$\widetilde{\alpha}_j(\Delta) = \underset{\forall t_{m,d,k}^{x,y};\delta(t_{m,d,k}^{x,y})=\Delta}{\operatorname{argmax}} \nu(t_{m,d,k}^{x,y}). \tag{6}$$

The infinite arrival curve α_j is then constructed as a periodic extension of $\widetilde{\alpha}_j$. Specifically, the periodic part is a scaled version of $\widetilde{\alpha}_j$, which is repeated l times for $l \in \mathbb{N}^+$ (infinitely). Therefore, α_j is defined as follows:

$$\alpha_j(\Delta) = \begin{cases} \widetilde{\alpha}_j(\Delta) & 0 \leq \Delta \leq w_j \\ \widetilde{\alpha}_j(\Delta - l \cdot w_j) + l \cdot \widetilde{\alpha}_j(w_j) & \text{otherwise } (l \in \mathbb{N}^+) \end{cases} \tag{7}$$

where $\widetilde{\alpha}_j(w_j) = \mu_j^{max,tot}$.

Fig. 3 and 4 depict the upper arrival curve for the superblock set S of Fig. 1 along with the tuples that were used for its derivation. For the arrival curve of Fig. 4, it has been given that the minimum gap that can exist between two successive executions of S is equal to $g = 40\mu$s, whereas for the arrival curve of Fig. 3 no similar information is available, so the minimum feasible gap has been conservatively assumed to be $g = 0$ in Eq. 4.

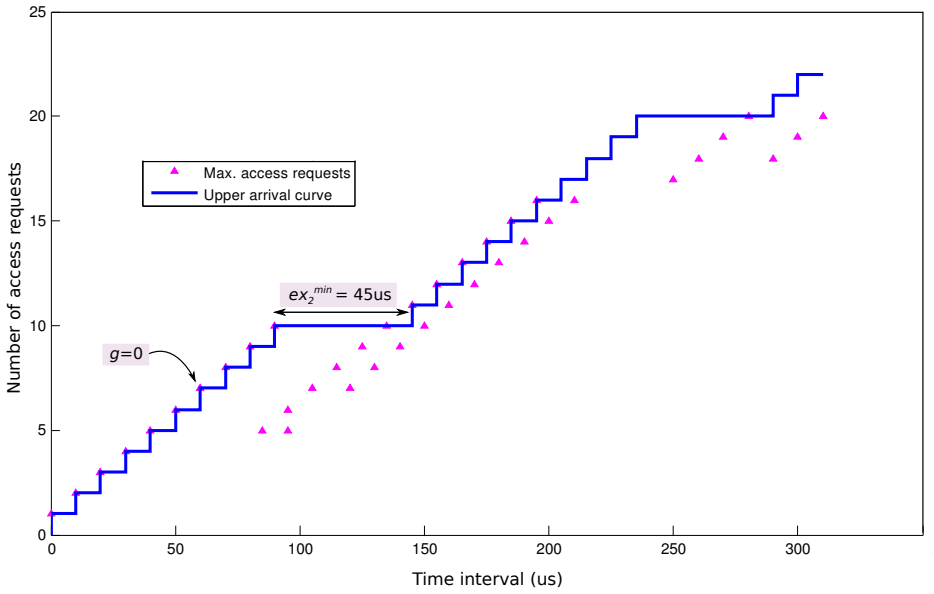

Figure 3: Upper arrival curve with constructing tuples for superblock set S ($C = 10\mu$s, $g = 0\mu$s)

2.4 Deriving the interference curve of multiple cores

When several processing cores (subset $R \subseteq P$) are considered, the sum of their individual arrival curves (which are derived as shown previously, by considering each core in isolation) represents a safe upper bound on the interference α that these cores might cause on the arbiter of the shared resource in any time interval Δ:

$$\alpha(\Delta) = \sum_{p_j \in R} \alpha_j(\Delta) \tag{8}$$

The *interference curve* α, which is computed by Eq. 8 is of particular importance for the WCRT analysis methodology of [4], since it enables the abstraction of task execution and resource accessing on several cores through a single arrival curve, which can easily modelled through a limited number of timed automata.

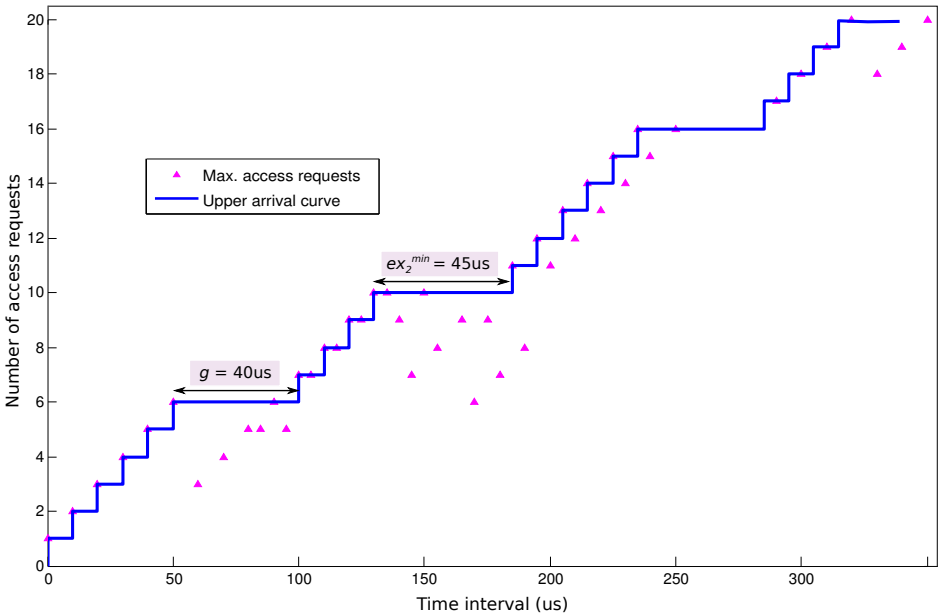

Figure 4: Upper arrival curve with constructing tuples for superblock set S ($C = 10\mu$s, $g = 40\mu$s)

3 Abstractions of Timed Automata-based System Specification

[4] presents in detail the network of timed automata that can be used to model precisely a multicore resource-sharing system, complying with the assumptions of Sec. 1. Because the application of model checking over such a system specification is often of prohibitively high complexity, the same work presents an abstraction combining timed automata with the real-time calculus, in particular using an arrival curve to represent the worst-case interference that several cores can cause over a shared resource (Sec. 2) and modeling this curve with a pair of timed automata. Besides this basic abstraction step, however, more abstractions and optimizations of the timed automata-based system specification can be considered, in order to reduce the complexity of model checking. The most important of them are discussed in the following.

3.1 System Specification with Timed Automata

1. Given that the complexity of model checking increases exponentially with the number of clocks and clock constants that a network of timed automata employs, it is crucial to avoid any redundant clock variables. For instance, if the system analysis is aimed at computing the WCRT of a

particular superblock within a core's processing cycle, it suffices to use the *Superblock* automaton (Fig. 2(a), Sec. 4.1.1 in [4]) only for the particular superblock of interest. All remaining superblocks, which are scheduled in the same processing cycle, can be modeled by a modified version of the *Superblock* automaton that does not employ clock `x` (`x` measures the elapsed time from the beginning to the end of a superblock execution). Another trivial optimization of the system specification can be applied when the processing cycle of a core features only one superblock or a superblock sequence of which the total WCRT is of interest. In this case, the *Scheduler* automaton for the particular core is redundant (its clock `p` measures the same elapsed time as clock `x` of *Superblock*) and hence, it can be omitted from the system specification.

2. In case of cores with synchronized processing cycles that access a FCFS- or RR- arbitrated resource, the state space exploration for the verification of a superblock's WCRT can be restricted to the duration of one hyper-period of the cores' processing cycles. The hyper-period is defined as the least common multiple of the cycles' periods ($lcm(W_1, \cdots, W_n)$) and within its duration all possible interference patterns are exhibited. Therefore, the derivation of a superblock's WCRT by exploring the feasible scenarios in this time window only ('pruned' state space) is safe.

To apply this optimization, the *Superblock* automaton has to be slightly modified, as shown in Fig. 5[4]. In particular, *Superblock* does no longer model the (infinite) periodic execution of a superblock, but a finite number of executions, given by the parameter `instances[id]` (occurences of superblock $s_{1,j}$ within the considered hyper-period, $\frac{lcm(W_1,\cdots,W_n)}{W_j}$).

It should be noted that a similar optimization can be also applied in case of TDMA- or FlexRay- arbitrated resources, on condition that the hyper-period is redefined to account also for the period of the arbitration cycle ($lcm(W_1, \cdots, W_n, \Theta)$, where Θ denotes the length of the respective arbitration cycle).

3. When verifying the WCRT of a superblock sequence on a core under analysis (CUA) with access to a TDMA-arbitrated resource, considering all possible interleavings between the scheduled phases on CUA and those on the remaining cores is not necessary. That is because of the timing isolation that such an arbitration scheme offers. Namely, one needs to consider all possible relative offsets between the arrival of CUA's access requests and the beginning of its dedicated slot in the arbitration cycle, independently of the activity on the remaining cores. Therefore, the *Superblock* and *Scheduler* automata for all cores other than CUA can be

[4]Note that the *Superblock* automaton of Fig. 5 employs clock `x` to measure the elapsed time within a period and it models superblocks which are fired at multiples of `period[id]`. This modification w.r.t. the basic *Superblock* automaton is applied to eliminate the need of a *Scheduler*, as suggested earlier.

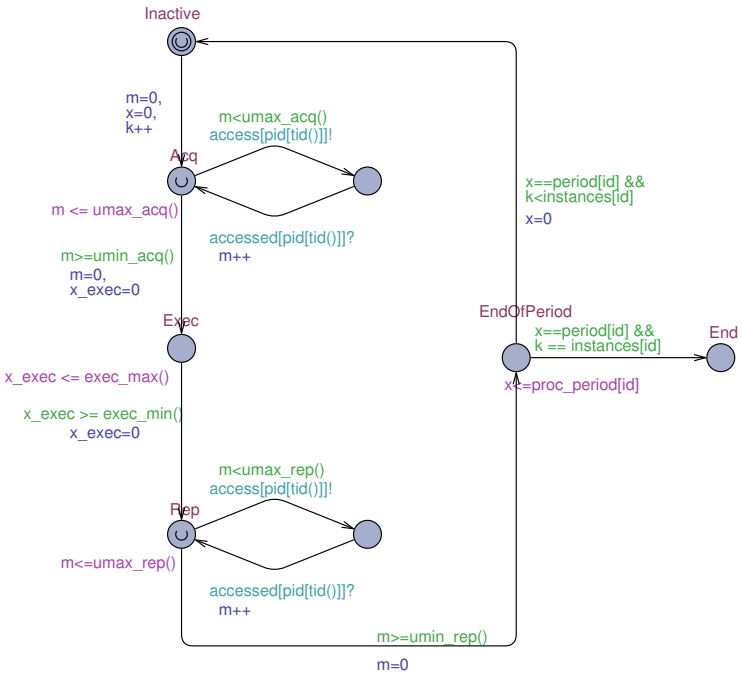

Figure 5: Superblock - `instances[id]` periodic executions with period `period[id]`.

completely eliminated from the system specification without affecting the correctness of the obtained WCRT estimates.

3.2 System Specification with Timed Automata and Real-Time Calculus

1. In a system specification where the interference caused by several cores is abstracted through their aggregate access request arrival curve α (Eq. 8, Sec. 2) and the shared resource is FCFS- or RR-arbitrated, the *Superblock* automaton for a superblock on the core under analysis can be simplified to model not periodic execution, but a single execution instead. Since all possible interference streams of the remaining cores (bounded by α) can be explored for the time interval of one superblock execution and due to the arrival curve property of sub-additivity, the WCRT observed during this interval ('pruned' state space) is a safe bound of the overall WCRT. This optimization also eliminates the need for representing the *Scheduler* and/or the remaining *Superblock* instances of the CUA, thus reducing further the number of timed automata, clocks and synchronization channels in the system specification.

The suggested optimization can be also applied for systems with a TDMA- or FlexRay-based resource arbiter. In this case, however, we

need to consider (enumerate and model) all possible offsets for the starting time of the superblock within the respective arbitration cycle, in order to obtain safe WCRT estimates. Consider, for example, that the analyzed superblock is executed for the first time at time 0 and that it is repeated periodically with a period of 1000μs, while the TDMA arbitration cycle has a period of 200μs, starting also from time 0. In this case, every new superblock execution coincides with the beginning of a new arbitration cycle (*offset*= 0). If the processing cycle had a period of 400μs though, every new superblock execution would start either at the beginning of an arbitration cycle or 200μs afterwards (*offset*$\in \{0, 200\}$). This information is modelled through variable `offset` in the modified *Superblock* automaton of Fig. 6. Verification of a superblock's WCRT has to be repeated for all its possible values, to account for all potential offsets between the emission of CUA's access requests and the beginning of its dedicated slot in the arbitration cycle.

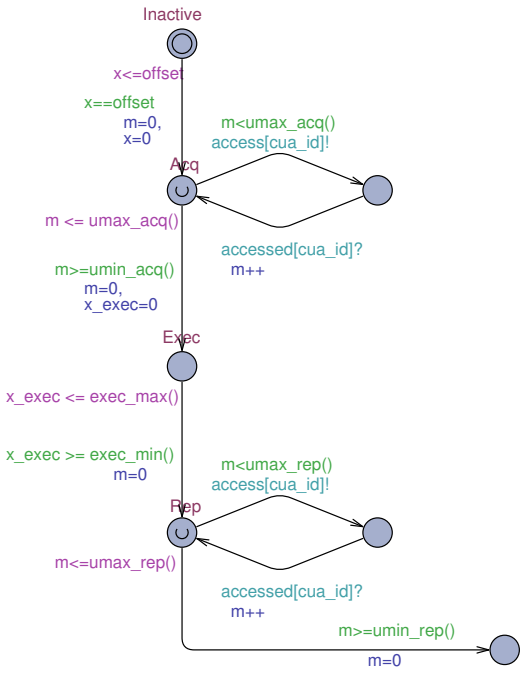

Figure 6: Superblock - Single execution starting `offset` time units after the beginning of the arbitration cycle.

2. For system specifications with a TDMA arbiter, the interference from the competing cores can be ignored (not modeled). This is because it does not affect the WCRT of the superblocks executing on CUA. The same holds also for the static segment of the FlexRay arbitration cycle.

3. In order to reduce the synchronisation frequency between the interference generating automata (*Upper Bound*, *Access Request Generator*, see

Sec. 4.2.2, [4]) and the RR or FlexRay *Arbiter*, it is possible to change the granularity of their communication, by enabling the interfering access requests to arrive in bursts at the arbiter. That means that the two interference generating automata can be slightly modified so that the access requests are no longer emitted one by one, but in bursts on b, where b is a divider of the maximum capacity B_{max} of the leaky bucket they implement. Similarly, the *Arbiter* automata have to be adapted to store the received access requests in their queues and serve them as they would in the original system (where they would have been emitted one by one). This optimization has no effect on the correctness or tightness of the WCRT estimates of CUA's superblock, while it causes a decrease in the verification time due to the reduced need for inter-automata synchronization.

4. The last optimization is related to the construction of the access requests arrival curves α_j for a system's processing cores. As has been already presented in Sec. 2, this requires information on the minimum 'gap' g between two successive executions of the superblock sequence S_j. If this information is not available, the gap is taken equal to 0 although this assumption may yield overly pessimistic arrival curves (see, e. g., Fig. 3). However, for systems with FCFS- or RR-arbitrated resources, the minimum gap can be conservatively estimated (even if no relevant information is provided) by assuming that every access of core p_j is delayed by all remaining cores. That is, we assume that every access request of p_j arrives at the resource arbiter immediately after a burst of (unserved) requests by all competing cores, which results in a response time $t_r = N \cdot C$ for every request. This pessimistic scenario provides a conservative WCRT estimate for S_j ($WCRT_{cons}(S_j) = \mu_j^{max,tot} \cdot t_r + \sum_{i=1}^{3\cdot|S_j|} ex_{i,j}^{max}$) and hence, a safe lower bound on g ($g = W_j - WCRT_{cons}(S_j)$), which can be used to construct a tighter arrival curve for p_j (see, e. g., Fig. 4). The suggested optimization does not reduce the complexity of model checking, but aims at reducing the pessimism of the WCRT estimates for CUA's superblocks.

4 Conclusion

This technical report presented (as a complement to [4]) methods to reduce the complexity of model checking for timed automata-based specifications of multicore systems, where tasks compete for access to mutually exclusive shared resources. As the case studies of [4] (Sec. 5) indicate, application of these methods (abstraction of interference of several cores with arrival curves and further optimizations based on arbitration policy of shared resource) can significantly boost the scalability of the suggested WCRT analysis method. This is a promising finding, opening new possibilities for tight and scalable interference analysis in multicore systems, which remain to be explored in

future work.

References

[1] Flexray communications system protocol specification, version 2.1, revision a. `http://www.flexray.com/`.

[2] R. Alur and D. L. Dill. Automata For Modeling Real-Time Systems. In *Automata, Languages and Programming*, pages 322–335. Springer, 1990.

[3] G. Behrmann, A. David, and K. G. Larsen. A tutorial on UPPAAL. In *Formal Methods for the Design of Real-Time Systems*, pages 200–236, 2004.

[4] G. Giannopoulou, K. Lampka, N. Stoimenov, and L. Thiele. Timed model checking with abstractions: Towards worst-case response time analysis in resource-sharing manycore systems. In *Proc. International Conference on Embedded Software (EMSOFT)*, Tampere, Finland, 2012.

[5] J.-Y. Le Boudec and P. Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*. Springer-Verlag, Berlin, Heidelberg, 2001.

[6] R. Pellizzoni, A. Schranzhofer, J.-J. Chen, M. Caccamo, and L. Thiele. Worst case delay analysis for memory interference in multicore systems. In *Design, Automation, Test in Europe Conference*, pages 741–746, 2010.

[7] A. Schranzhofer, J.-J. Chen, and L. Thiele. Timing analysis for TDMA arbitration in resource sharing systems. In *Real-Time and Embedded Technology and Applications Symposium*, pages 215–224, 2010.

[8] A. Schranzhofer, R. Pellizzoni, J.-J. Chen, L. Thiele, and M. Caccamo. Worst-case response time analysis of resource access models in multicore systems. In *Design Automation Conference*, pages 332–337, 2010.

[9] A. Schranzhofer, R. Pellizzoni, J.-J. Chen, L. Thiele, and M. Caccamo. Timing analysis for resource access interference on adaptive resource arbiters. In *Real-Time and Embedded Technology and Applications Symposium*, pages 213–222, 2011.

[10] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *Symposium on Circuits and Systems*, volume 4, pages 101–104, 2000.

[11] R. Wilhelm, D. Grund, J. Reineke, M. Schlickling, M. Pister, and C. Ferdinand. Memory hierarchies, pipelines, and buses for future architectures in time-critical embedded systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(7):966 –978, 2009.