Response-Time Analysis of Arbitrarily Activated Tasks in Multiprocessor Systems with Shared Resources

Mircea Negrean, Simon Schliecker, Rolf Ernst
Institute of Computer and Communication Network Engineering
Technical University of Braunschweig
D-38106 Braunschweig / Germany
{negrean|schliecker|ernst}@ida.ing.tu-bs.de

Abstract—**As multiprocessor systems are increasingly used in real-time environments, scheduling and synchronization analysis of these platforms receive growing attention. However, most known schedulability tests lack a general applicability. Common constraints are a periodic or sporadic task activation pattern, with deadlines no larger than the period, or no support for shared resource arbitration, which is frequently required for embedded systems. In this paper, we address these constraints and present a general analysis which allows the calculation of response times for fixed priority task sets with arbitrary activations and deadlines in a partitioned multiprocessor system with shared resources. Furthermore, we derive an improved bound on the blocking time in this setup for the case where the shared resources are protected according to the Multiprocessor Priority Ceiling Protocol (MPCP).**

I. Introduction

The design and development of multiprocessor systems has become a hot topic for most major semiconductor companies in recent years. Driven by the increasing performance requirements and power limitation of single processor systems, different multiprocessor and multi-core solutions have been already provided not only for general-purpose processors but also for embedded devices [1], [2]. As such components are often used in real-time applications, formal verification of the timing properties is essential.

Timing is however not trivial in multiprocessor systems. Even in setups with static task-to-processor mapping, the execution of the tasks is usually not independent. The use of the same physical hardware, such as memories, coprocessors, or network components, makes inter-core interference unavoidable and may introduce hard-to-find timing problems including missed deadlines that can finally make the entire system fail. Thus, the analysis of tasks on multiprocessor systems becomes more complicated under synchronization mechanisms. Effective solutions to handle resource sharing have been suggested for uniprocessor systems, e.g. [3] and based on this solution different extensions and variations for multiprocessor systems were proposed, e.g. [3], [4].

To provide the necessary timing guarantees for such systems, various formal scheduling analysis techniques have been proposed that cover partitioned and non-partitioned multiprocessor scheduling with varying degrees of generality. For long, however, real-time analysis has focused on constrained task models — i.e. periodic activation and deadlines not larger than the periods. Moreover, synchronization of shared resources, essential elements of every real-time system, are often neglected. While both issues have individually been addressed in isolation, no method is currently available that allows the verification of real-time systems that exhibit both of these properties.

The contribution of this paper is the response time analysis for multiprocessor partitioned real-time systems composed of a set of tasks with arbitrary activations and shared resources. With this, we overcome the restriction of previous work which assumes that task deadlines are less than or equal to the activation period or does not consider synchronization issues. The partitioning problem is not our concern in this paper, but the improved schedulability test can be used to allow more efficient partitioning.

The remainder of this paper is organized as follows. First, we provide an overview of related work in Section II. The system model we use is introduced in Section III. The new response time analysis is presented in Section IV. Section V provides an example, underlining the applicability of the proposed approach. We draw conclusions in Section VI.

II. Related work

Multiprocessor scheduling algorithms are an intensively investigated topic in the real-time community. Two major categories of multiprocessor scheduling can be identified: the *partitioned* and the *global / non-partitioned* approach. Feasibility tests are available for a variety of partitioned and global scheduling policies such as static priority preemptive [5], [6] or global EDF [7]. Arbitrary sporadic task systems on preemptive multiprocessor systems under the partitioned paradigm were investigated in [8]. In [9] the authors propose a schedulability test for multiprocessor systems based on response time analysis. They extend the previously known response time analysis technique from the uniprocessor scheduling theory to globally scheduled periodic and constrained sporadic tasks.

978-3-9810801-5-5/DATE09 © 2009 EDAA

Under partitioned scheduling, each task is statically mapped to a processor, e.g. according to bin-packing heuristics, and then schedulability tests are used to verify whether all tasks meet their deadlines. Since the partitioning of tasks among processors separates the multiprocessor scheduling problem into multiple uniprocessor scheduling problems, well developed uniprocessor scheduling techniques can be directly applied. Many of these analyses are based on a variation of the busy window technique proposed by Lehoczky [10] and extended by Tindell et. al [11].

The classical assumption for real-time tasks in scheduling algorithms is that they are independent. In practice however, tasks interact via semaphores to synchronize the shared resources between different processors. Rajkumar et al. [3] propose a locking protocol for real-time multiprocessor systems known as the Multiprocessor Priority Ceiling Protocol (MPCP). This protocol assumes that tasks are scheduled according to the partitioned rate monotonic scheduling (RMS) policy and allows to bound the time a task is delayed by other local or remote tasks due to resource contention. They propose a schedulability condition that needs to be checked for each processor. In [4] Chen et al. propose a modified resource control protocol that is similar to MPCP but can be used together with partitioned EDF or partitioned RMS. Another shared resource arbitration algorithm is the Multiprocessor Stack Resource Protocol (MSRP) which is compared with MPCP in [12], showing that neither outperforms the other on the complete spectrum of system setups. In [13] Devi et al. present a synchronization procedure specifically for multiprocessor scheduling under global EDF.

In [14] an end-to-end approach to derive worst case response time bounds for tasks mapped on multiprocessor systems and sharing resources is presented.

All of the approaches considering synchronization aspects on multiprocessor systems assume a periodic or a classical sporadic task model. More general task activation patterns can be expressed with event streams (see Fig. 1), which can be described using the upper and lower event arrival functions $\eta^+(\Delta t)$ and $\eta^-(\Delta t)$. These specify the maximum respectively the minimum number of events that occur in the event stream during any time interval of length Δt.

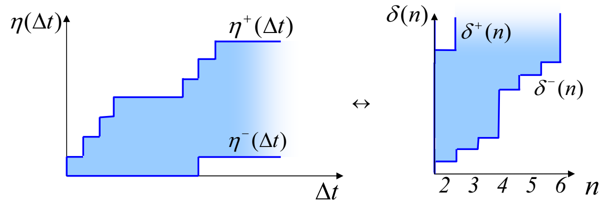

Fig. 1. Event Streams.

Correspondingly, an event stream can also be specified using the functions $\delta^-(n)$ and $\delta^+(n)$ that represent the minimum and respectively the maximum distance between any n ($n \geq 2$) events in the stream. Compact models have been suggested to efficiently cover common event stream properties (such as periodicity or event jitter) [15], [16], [17].

In event driven multiprocessor systems, task activations are often the result of messages being sent from other tasks. Here, the derivation of the task activating event model is not straight-forward. As a solution, compositional scheduling analysis approaches [16], [17] propose to propagate the event stream information during the analysis. Iteration is used where cyclic dependencies exist.

III. System model

For the scope of this paper, we assume a multiprocessor real-time system composed of a set of m processors, each having its own static priority preemptive scheduler, a set of shared resources, and a static set of n arbitrarily activated tasks $\tau = \{\tau_1, \dots \tau_n\}$. The tasks have been statically mapped to the processors with some method, and it is our goal to determine the schedulabilty of the given mapping. A common priority space across all processors is assumed and each task in the system has a unique, static priority. Each instance of a task, called a job, is activated by events, which are modeled as arbitrary event streams using solely the functions $\eta^+(\Delta t)$, $\eta^-(\Delta t)$, $\delta^+(n)$ and $\delta^-(n)$. Each job of a task τ_i executes no longer than a worst case execution time C_i, and must complete before a given (relative) deadline D_i, which may be smaller, equal, or larger than the distance to the successive activation. Thus, if a task has a worst case response time larger than this distance, it is possible for that task to re-arrive before the previous job has completed. We assume that new jobs may not start to execute before the previous job is complete.

Resources are objects that require serialized access. We distinguish between local and global resources. Local resources reside on each processor and can be accessed only by tasks mapped on it. Global resources are assumed in a separate shared resource module and can be accessed by tasks mapped on different processors.

IV. Schedulability analysis

As an example, consider a multiprocessor system consisting of two cores as shown in Fig. 2. An independent static priority preemptive operating system is executing on each core.

The worst case response time of a task mapped on any of these processors is classically determined by the task's worst case execution time plus the maximum amount of time the task can be kept from executing due to preemptions by higher priority local tasks. Furthermore, when a task performs accesses to shared resources it is additionally delayed (blocked) when waiting for the required resources. While in uniprocessor systems this delay depends only on tasks mapped on the same processor, in multiprocessor systems it also depends on the amount of load imposed on the shared resources by tasks mapped on other processors in the system. Thus, the local analysis of one processor now depends on the shared resource interference caused by other processors.

To calculate the task response times in the given setting, we need to address three problems. First, the load imposed by tasks on shared resources has to be determined. Second, this

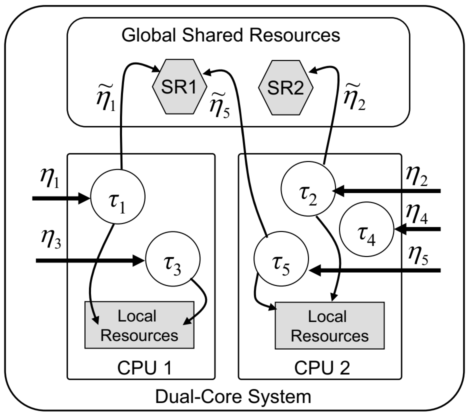

Fig. 2. Dual-Core system with tasks accessing local and global resources.

information has to be used to derive the maximum blocking time that a task may experience. Third, the obtained blocking times need to be integrated in the worst case response time. This step couples local scheduling analysis with the analysis of the shared resource arbitration. These issues are addressed in the following sections.

A. Derivation of shared resource load

A limitation of previous approaches to derive the blocking time in multiprocessor system is the relatively simple model of the task's individual resource usage. For example, [3] assumes a constant number of requests per task execution. In single processor systems, the blocking time can be precisely calculated without knowing the task's exact request pattern. This is due to the fact that only a single critical section can block a higher priority task, making the exact request timings insignificant. This is not true for multiprocessor systems. Here, various sources of blocking from remote processors may occur, and requests from different processors may be prioritized, causing several "blockings" by the same task. It is therefore advisable, to take a closer look at how the requests are timed.

Imagine a task τ_1 that is trying to access a global shared resource SR that is also used by another task τ_2 on another processor as depicted in Fig. 3. The resource arbitration is such that τ_2 receives a higher priority on the resource, thus conflicts are resolved in its favor. Now, assume τ_1 tries to access the resource 4 times during its execution. The same is true for τ_2. The exact timing of τ_2's accesses now clearly makes a difference. If all requests occur at the beginning of its execution (Fig. 3a), this may cause τ_1 to be blocked each time. If however, τ_2's requests are further separated in time (Fig. 3b), this means that during τ_1's execution, only one or two conflicts may actually occur.

To capture such details for the analysis, we apply the event model concept which has previously been used to model task activations ([16], [17]) to also capture the resource traffic. For this we define:

Definition 1. *The* Shared Resource Request Bound $\tilde{\eta}_i^+(\Delta t)$ *is the maximum number of requests that may be issued by a task τ_i to a shared resource within a time window of size Δt.*

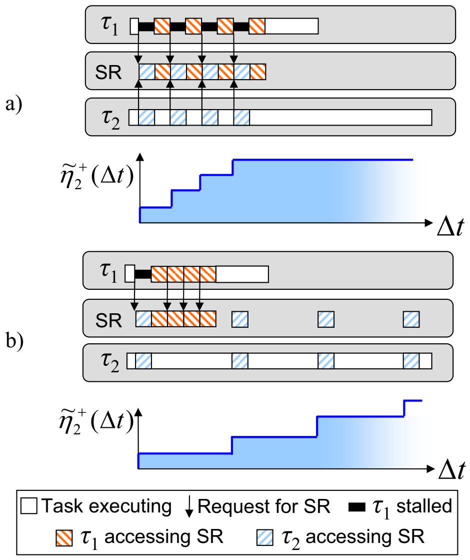

Fig. 3. Effect of Resource Load Variations.

This model provides the opportunity to cover several new aspects in the analysis of shared resource timing. Firstly, it allows to express the inter event timing between several requests by one job of the task, which leads to more accurate analysis results (as shown in Section V). Secondly, it can be used to conveniently capture the timing of the joint traffic by several different jobs, and even several different tasks. This allows simplifying the analysis of shared resource timing as shown in the next section.

The shared resource request bound of a task τ_i can be straight-forwardly bounded: Let τ_i be activated by events bounded by event model $\eta_i^+(\Delta t)$, have a worst case response time R_i, and perform at most n_i accesses to a shared resource per activation. The request bound of a task is then given by

$$\tilde{\eta}_i^+(\Delta t) = \eta_i^+(\Delta t + R_i) \cdot n_i. \tag{1}$$

Note that (1) features the η_i^+-function shifted by the task's response time to account for the requests of jobs that are unfinished at the beginning of the investigated time interval. The joint shared resource request bound of multiple tasks can then be calculated as the sum over the individual request bounds of each task.

This simple model does not yet consider the distances between events as in Fig. 3b. A more accurate shared resource request bound can be derived by measurement, or, more reliably, by analysis of the task's internal control flow (i.e. its linear execution sequences, jumps, and conditional statements) and its external activation pattern. For example, a task that makes an access to a shared resource within a loop, will produce a request sequence that contains several accesses (one per loop) separated by the loop execution time, and the overall pattern repeating with each activation of the task.

Further details on deriving $\tilde{\eta}(\Delta t)$ are beyond the scope of this paper. The interested reader is referred to similar approaches for deriving a bound on the number of memory accesses [18], [19].

B. *Derivation of blocking times*

In the following we will consider the *multiprocessor priority ceiling protocol* (MPCP) [3] and using the load derivation on the shared resources we will introduce an improved blocking time analysis for task sets activated by the general event stream model.

Since task deadlines can be larger than their periods, the blocking time analysis has to consider the possible influence of overlapping job execution. This influence can be captured by analysing the tasks during their execution in a time interval denoted here with $w_i(q)$ — q ($q = 1, 2, \dots k$) represents the number of activations of a task τ_i within $w_i(q)$. Further details about the computation of $w_i(q)$ will be introduced in Section IV-C. Thus, the blocking time B_i of a task τ_i when accessing local and global resources in a multiprocessor system can be determined as a function of the size of a time window $w_i(q)$ during which task τ_i executes.

MPCP is a deadlock free protocol which relies on the following assumptions: a task τ_i can access local or global resources; a critical section guarded by a semaphore and protecting a global or a local resource is called global critical section (gcs) or local critical section (lcs); priority ceilings are assigned to critical sections; local critical sections are assigned priority ceilings according to the uniprocessor PCP; global critical sections are assigned priority ceilings that are higher than the priority of any other task in the system; during execution, tasks are suspended when they try to access a locked gcs; when a higher priority task is blocked on a global critical section local tasks can be executed and may even try a lock on local or global critical sections; global critical sections are not allowed to be nested in other critical sections (local or global) and vice-versa; if tasks perform nested accesses to global critical sections, an explicit partial ordering of global resources has to be used to prevent deadlocks.

Rajkumar [3] identified that the blocking time of a task τ_i due to resource contention in a multiprocessor system consists of up to 5 types of blocking. In order to present the blocking factors, the following definitions are introduced, which are similar to the ones used in [3].

- The maximum number of global critical sections that each job J_i of a task τ_i executes before its completion is n_i^G.
- $\tilde{\eta}_j^+(w_i(q))$ represents the maximum number of requests that any task τ_j can issue to a shared resource within the investigated time interval $w_i(q)$.
- ω_i^{local} and ω_i^{global} represent the maximum duration of a local and respectively of a global critical section when it is accessed by jobs of a task τ_i.
- $lp(i)$ is the set of tasks with lower priority than τ_i on its processor.
- We denote the set of lower priority local tasks that require global resources with $lp(i)^G$.
- $lpr(i)$ and $hpr(i)$ are the sets of tasks which are mapped on remote processors and have lower and respectively higher priority than τ_i.
- $GS_{i,j}$ represents the set of global semaphores that will be locked by jobs of both tasks τ_i and τ_j.
- The set of tasks which are elements of $lpr(i)$ and access elements of $GS_{i,j}$ is denoted with $\theta_{i,j}$.
- Similar the set of tasks which are elements of $hpr(i)$ and access elements of $GS_{i,j}$ is denoted with $\Theta_{i,j}$.
- Jobs of the tasks in $\theta_{i,j}$ and $\Theta_{i,j}$ are jobs which directly block jobs of task τ_i. Consider the processors on which tasks in $\theta_{i,j}$ and $\Theta_{i,j}$ are mapped. Each of these processors may contain other tasks that access global resources with higher priority ceilings than the priority ceiling of the resources accessed by tasks directly blocking τ_i. We denote the set of these tasks with $\Psi_{i,j}$.

Based on these definitions, we now extend the 5 blocking factors of the classical MPCP analysis to consider the influence of multiple job activations and the load imposed on the shared resources.

Local blocking time. According to the uniprocessor priority ceiling protocol (PCP), each job J_i of a task τ_i may be blocked once by a job J_j of a lower priority local task $\tau_j \in lp(i)$. In the occurrence of overlapping activations of task τ_i, a lower priority local job J_j will block only the first job of the task τ_i (once J_j exits the critical section which blocks J_i it cannot execute anymore before all jobs of τ_i are finished). Additionally, in the multiprocessor protocol, each time a job J_i tries to lock a global semaphore, it can potentially suspend, letting lower priority jobs execute on the local processor. This reproduces the situation presented above where jobs of lower priority tasks can lock local resources each time J_i attempts to enter a global critical section and suspends. These low priority jobs can lock local semaphores and block J_i when it resumes its execution. Therefore, the local blocking time of a job J_i can be bounded by:

$$B_{i1}(w_i(q)) = [1 + q \cdot n_i^G] \cdot \max_{\forall \tau_j \in lp(i)} (\omega_j^{local})$$

Direct blocking time. Each time a job J_i tries to access a global critical section, it can find that this is currently held by a lower priority job on a different processor. Thus, the blocking time due to lower priority remote tasks which share the same global resources with J_i (jobs of tasks in the set $\theta_{i,j}$) is:

$$B_{i2}(w_i(q)) = q \cdot n_i^G \cdot \max_{\forall \tau_j \in \theta_{i,j}} (\omega_j^{global})$$

Similar, each job J_i can be blocked by higher priority remote jobs that request the same global resource as J_i (jobs of tasks in the set $\Theta_{i,j}$). As opposed to lower priority remote jobs, higher priority remote jobs may be served multiple times.

$$B_{i3}(w_i(q)) = \sum_{\forall \tau_j \in \Theta_{i,j}} (\tilde{\eta}_j^+(w_i(q)) \cdot \omega_j^{global})$$

Indirect preemption delay. We now consider the processors on which tasks that directly block jobs of task τ_i (tasks in $\theta_{i,j}$ and $\Theta_{i,j}$) are mapped. If tasks on these processors (tasks in $\Psi_{i,j}$) access global resources with higher priority ceilings than the priority ceilings of the resources accessed by tasks directly blocking τ_i, each of them can preempt the global critical sections of tasks directly blocking τ_i. Their influence on the blocking time can be captured by:

$$B_{i4}(w_i(q)) = \sum_{\forall \tau_j \in \Psi_{i,j}} (\tilde{\eta}_j^+(w_i(q)) \cdot \omega_j^{global})$$

Local preemption delay. Each time a job J_i of task τ_i tries to access a global resource, it can potentially suspend, letting jobs of lower priority local tasks execute on its local processor. If these jobs require access to global resources (jobs of tasks $\tau_j \in lp(i)^G$), they can lock or queue up on the global resources and can therefore preempt J_i when it executes non-critical code. Within the investigated time interval $w_i(q)$ there are at most q jobs of task τ_i and each of these jobs can issue maximal n_i^G requests to global resources. In addition, when J_i begins its execution on its local processor, a lower priority job can have an outstanding request for a global semaphore. Hence, in the analyzed time interval, task τ_i can be blocked for at most $q \cdot n_i^G + 1$ global critical sections of tasks in $lp(i)^G$. But, lower priority local tasks that require access to global resources can issue at most $\tilde{\eta}_j^+(w_i(q))$ requests to global resources within $w_i(q)$. As a result, only the minimum of these two bounds may actually occur.

$$B_{i5}(w_i(q)) = \sum_{\forall \tau_j \in lp(i)^G} \min(q \cdot n_i^G + 1, \tilde{\eta}_j^+(w_i(q))) \cdot \omega_j^{global}$$

The worst case blocking time that a job of a task τ_i can encounter in a time window $w_i(q)$ is given by the sum of the above 5 blocking factors ($B_{i1}, \dots B_{i5}$).

C. Multiprocessor response time analysis

In this section, we provide the schedulability condition for tasks under the partitioned multiprocessor static priority preemptive scheduling with shared resources and arbitrary task activations. For this, we extend the classical busy window approach for uniprocessors by Tindell et. al [11].

In a uniprocessor system under static priority preemptive scheduling, the response time of a task τ_i is given by the largest response time of any of the q ($q = 1, 2, \dots k$) task activations that lie within the busy interval. The response time of the q-th activation of task τ_i is given by the difference between the window length $w_i(q)$ and the moment when this activation was initiated relative to the beginning of the busy interval. This is given by $\delta_i^-(q)$. Thus, $R_i = \max(w_i(q) - \delta_i^-(q))$.

Two important aspects need to be considered to extend Tindell's formula. Firstly, we can not rely on the critical instance scenario anymore, because the use of globally shared resources can lead to suspension of tasks, which possibly defers the task execution times (see also [3]). The busy window consists not only of the time interval during which task τ_i or a higher priority task is executing, but more generally the time interval during which at least one invocation of τ_i is not finished. This leads to an increased interference for τ_i, which includes also unfinished invocations of τ_j that have started before the investigated busy window. This is covered by shifting τ_j's activation function $\eta_j^+(\Delta t)$ by its worst case response time R_j. Secondly, the blocking time $B_i(w_i(q))$ is in our case a function of the window size during which the requests are issued (a larger time window can draw increased remote blocking, mainly from B_{i3} and B_{i4}). Thus, the worst case response times of tasks on partitioned multiprocessor systems can be calculated with (2).

$$w_i(q) = q \cdot C_i + \sum_{\forall \tau_j \in hp(i)} \eta_j^+(w_i(q) + R_j) \cdot C_j + B_i(w_i(q)) \quad (2)$$

where $w_i(q)$ is the maximum busy window of q activations of task τ_i; C_i is the worst case execution time of τ_i; $hp(i)$ is the set of tasks with higher priority than τ_i; $\eta_j^+(w_i(q) + R_j)$ is the maximum amount of jobs of τ_j in a time window of size $w_i(q)$; and $B_i(w_i(q))$ is the maximum blocking time computed as presented in the previous section.

A solution can be computed iteratively, because all components grow monotonically with respect to the window size. The response time has been found, when two successive iterations provide identical results. Finally, the schedulability test consists of checking whether the condition $R_i \le D_i$ holds for every task τ_i in the system.

Note that the response times of tasks in a multiprocessor system with voluntary suspension can not be calculated in an arbitrary sequence, because (2) requires the knowledge of the response time of higher priority tasks. To tackle this dependency the response times can be calculated top-down, starting with the highest-priority task. In addition, blocking factors B_{i3}, B_{i4}, ands B_{i5} may rely on the resource request bound (and indirectly the response time) of *lower* priority tasks, which leads to a cyclic dependency. In [3], this problem is tackled with an extension of the resource arbitration protocol by a so called *period enforcer*. Alternatively, instead of using the request bound (1), our approach allows the use of bounds that are independent of the task's response time (e.g. based on the fact that job executions of the same task may not overlap or that the requests are separated by a minimum distance).

In contrast to previous work, such as [3], our approach does not require time-triggered task activations, but is conceived to be appropriate also for event-driven multiprocessor systems. This is possible through the consistent use of generic event models to describe task activations and shared resource requests. A major issue here is, that a task's activating event model may not initially be known — in particular when it is the effect of e.g. another task finishing or data arriving over a bus. This problem is addressed by embedding the analysis into a compositional analysis approach such as [17] and [20], where task activating event models are provided and iteratively refined during the analysis procedure. The analysis of the shared resource bounds and the above response time analysis then need to be repeated as well, until the definite event models have been found.

V. Experiments

To evaluate our analysis, we compare it with the analysis presented in [3]. For this, we assume that the system depicted in Fig. 2 is stimulated purely periodically with the parameters given in Table I. In this setting both analyses compute τ_5's response time with $R_5 = 94$.

TABLE I
PARAMETERS FOR EXAMPLE SYSTEM

Task Name	Period T_i (ms)	Core Execution Time C_i (ms)	Global Resource Accesses $n_i^G * \omega_i^G$	Local Resource Accesses $n_i^L * \omega_i^L$
τ_1	1000	[50,300]	7 * 2	1 * 2
τ_3	500	[30,30]	-	1 * 5
τ_2	200	[10,10]	1 * 2	1 * 2
τ_4	50	[10,10]	-	-
τ_5	250	[40,40]	3 * 2	1 * 2

Based on this, we conduct a series of experiments in which we investigate the system performance with additional knowledge of shared resource request timing. For modeling purposes we assume that the shared resource request bound of τ_1 is given by the simple function $\tilde{\eta}_1(\Delta t) = \lceil \Delta t / d_{srr} \rceil$, thus all request by τ_1, representing direct remote blocking, are separated by a minimum distance of d_{srr}. As can be seen in Fig. 4a, the larger this distance becomes, the lower is the load on the shared resource. This directly allows τ_5 to finish faster, and indirectly, the faster execution draws less local interference, causing an over-proportional benefit for τ_5's response times (as seen at $d_{srr} = 27$). With increasing request distances, also the benefit of using our approach increases, being about 20% more accurate for $d_{srr} = 40$.

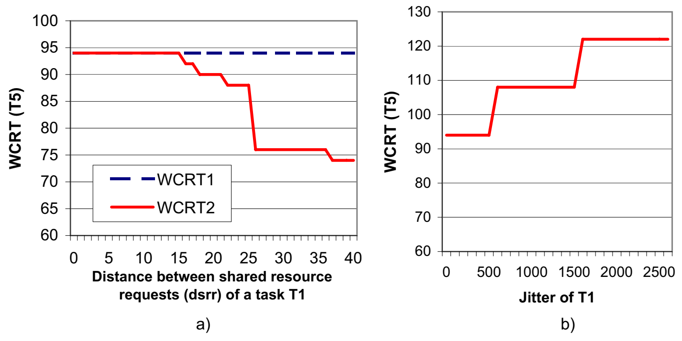

Fig. 4. Influence of Different Parameters. (WCRT1: response times calculated with the analysis approach for MPCP in [3]; WCRT2: response times calculated with the improved analysis approach for MPCP).

In the final experiment, we deviate from the periodic assumptions. For this, we increase the jitter of the event stream model for τ_1, which captures transient overload situations. The effect of the resulting increased interference can be seen in Fig. 4b, where the worst case response time of task τ_5 on the other processor is negatively affected. Depending on the given deadline, the system may even become unschedulable.

VI. CONCLUSION

In this paper we have provided a response time analysis for partitioned multiprocessor real-time systems composed of a set of tasks with fixed priorities that are arbitrarily activated and share secondary resources. This overcomes the restrictions of the majority of previous work, which either assumes that tasks' deadlines are less than or equal to the activation period or assumes tasks to be independent. We have provided tight response time bounds that allow an efficient partitioning of tasks in multiprocessor systems, increasing the effective utilization. Not only did we extend the scope of allowed system configurations, but we also improved the results of the traditional analyses for MPCP by utilizing improved resource usage models.

REFERENCES

[1] "The cell project at IBM research. http://www.research.ibm.com/cell/."

[2] "Freescale customers case studies: Continental - high-performance MCU optimized for EBS applications. http://www.freescale.com/."

[3] R. Rajkumar, *Synchronization in Real-Time Systems: A Priority Inheritance Approach.* Kluwer Academic Publishers Norwell, MA, USA, 1991.

[4] C.-M. Chen and S. K. Tripathi, "Multiprocessor priority ceiling based protocols," University of Marylands, Tech. Rep., 1994.

[5] B. Andersson, S. Baruah, and J. Jonsson, "Static-priority scheduling on multiprocessors," *Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS)*, pp. 193–202, Dec. 2001.

[6] B. Andersson and J. Jonsson, "The utilization bounds of partitioned and pfair static-priority scheduling on multiprocessors are 50%," *Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS)*, pp. 33–40, July 2003.

[7] S. Baruah and T. Baker, "Global EDF schedulability analysis of arbitrary sporadic task systems," *Euromicro Conference on Real-Time Systems (ECRTS)*, pp. 3–12, July 2008.

[8] S. K. Baruah and N. W. Fisher, "The partitioned dynamic-priority scheduling of sporadic task systems," *Real-Time Systems*, vol. 36, no. 3, pp. 199–226, 2007.

[9] M. Bertogna and M. Cirinei, "Response-time analysis for globally scheduled symmetric multiprocessor platforms," *28th IEEE International Real-Time Systems Symposium (RTSS)*, pp. 149–160, Dec. 2007.

[10] J. Lehoczky, "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines," *Proceedings 11th Real-Time Systems Symposium*, pp. 201–209, Dec 1990.

[11] K. W. Tindell, A. Burns, and A. J. Wellings, "An extendible approach for analyzing fixed priority hard real-time tasks," *Real-Time Syst.*, vol. 6, no. 2, pp. 133–151, 1994.

[12] P. Gai, M. Di Natale, G. Lipari, A. Ferrari, C. Gabellini, and P. Marceca, "A comparison of MPCP and MSRP when sharing resources in the Janus multiple-processor on a chip platform," *Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 189–198, May 2003.

[13] U. Devi, H. Leontyev, and J. Anderson, "Efficient synchronization under global EDF scheduling on multiprocessors," *Proceedings of the 18th Euromicro Conference on Real-Time Systems (ECRTS)*, pp. 75–84, 2006.

[14] J. Sun, R. Bettati, and J.-S. Liu, "An end-to-end approach to schedule tasks with shared resources in multiprocessor systems," *Proceedings., 11th IEEE Workshop on Real-Time Operating Systems and Software*, pp. 18 – 22, May 1994.

[15] K. Gresser, "An event model for deadline verification of hard real-time systems," in *Proceedings 5th Euromicro Workshop on Real-Time Systems*, Oulu, Finland, 1993, pp. 118–123.

[16] S. Chakraborty, S. Künzli, and L. Thiele, "A general framework for analysing system properties in platform-based embedded system designs," *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pp. 190–195, 2003.

[17] K. Richter, "Compositional scheduling analysis using standard event models," Ph.D. dissertation, Technical University of Braunschweig, 2004.

[18] S. Schliecker, M. Ivers, and R. Ernst, "Memory Access Patterns for the Analysis of MPSoCs," *IEEE North-East Workshop on Circuits and Systems*, pp. 249–252, 2006.

[19] K. Albers, F. Bodmann, and F. Slomka, "Hierarchical Event Streams and Event Dependency Graphs: A New Computational Model for Embedded Real-Time Systems," *Proceedings of the 18th Euromicro Conference on Real-Time Systems (ECRTS)*, pp. 97–106, 2006.

[20] S. Schliecker, J. Rox, M. Negrean, K. Richter, M. Jersak, and R. Ernst, "System level performance analysis for real-time automotive multicore and network architectures." *IEEE Transactions on Computer Aided Design*, 2009, (to appear).