Bounding Worst-Case Response Time for Tasks With Non-Preemptive Regions *

Harini Ramaprasad, Frank Mueller
Dept. of Computer Science, Center for Efficient, Secure and Reliable Computing
North Carolina State University
Raleigh, NC 27695-8206, mueller@cs.ncsu.edu

Abstract

Real-time schedulability theory requires a priori *knowledge of the worst-case execution time (WCET) of every task in the system. Fundamental to the calculation of WCET is a scheduling policy that determines priorities among tasks. Such policies can be non-preemptive or preemptive. While the former reduces analysis complexity and overhead in implementation, the latter provides increased flexibility in terms of schedulability for higher utilizations of arbitrary task sets. In practice, tasks often have non-preemptive regions but are otherwise scheduled preemptively. To bound the WCET of tasks, architectural features have to be considered in the context of a scheduling scheme. In particular, preemption affects caches, which can be modeled by bounding the cache-related preemption delay (CRPD) of a task.*

In this paper, we propose a framework that provides safe and tight bounds of the data-cache related preemption delay (D-CRPD), the WCET and the worst-case response times, not just for homogeneous tasks under fully preemptive or fully non-preemptive systems, but for tasks with a non-preemptive region. By retaining the option of preemption where legal, task sets become schedulable that might otherwise not be. Yet, by requiring a region within a task to be non-preemptive, correctness is ensured in terms of arbitration of access to shared resources. Experimental results confirm an increase in schedulability of a task set with non-preemptive regions over an equivalent task set where only those tasks with non-preemptive regions are scheduled non-preemptively altogether. Quantitative results further indicate that D-CRPD bounds and response-time bounds comparable to task sets with fully non-preemptive tasks can be retained in the presence of short non-preemptive regions.

To the best of our knowledge, this is the first framework that performs D-CRPD calculations in a system for tasks with a non-preemptive region.

* This work was supported in part by NSF grants CCR-0310860, CCR-0312695 and CNS-0720496.

1. Introduction

Bounding the worst-case execution times of tasks *a priori* is a requirement of schedulability analysis in hard real-time systems and an area of research that has received significant attention over many years. Several modern architectural features increase the complexity of the analysis to determine these WCET bounds statically by making execution behavior more unpredictable. A data cache is one such feature that is particularly hard to analyze. Providing *exact* WCET estimates using static analysis is a very hard problem, which generally results in unacceptable analysis complexity and overhead. Hence, all existing tools provide safe upper bounds.

Analyzing data cache behavior for single tasks is challenging in itself and has been the focus of much research for many years . However, this is not sufficient since systems usually have multiple tasks that execute in a prioritized, preemptive environment.

In prior work, we proposed a framework to provide worst-case response time estimates for tasks in a multi-task, preemptive, hard real-time environment [14, 15]. In such a system, every task has a priority. A task with a higher priority may preempt a task with a lower priority. The lower priority task then experiences a data-cache related preemption delay (D-CRPD) when it resumes execution, which increases its WCET and, hence, response time.

A fundamental assumption in our previous analysis is that all tasks be completely preemptive. In other words, a task may be interrupted by a task with higher priority *at any time* during its execution. This assumption may not be valid for some tasks. A task may have a period in its execution during which it performs some critical operations and, if interrupted, could produce incorrect results.

In our current work, we relax this assumption and allow tasks to have a region within their execution where they may not be preempted. We call this the non-preemptive region (NPR) of a task. We propose a framework that statically analyzes task sets within such an environment and calculates worst-case response times for all tasks.

The complexity of our analysis arises from the fact that the actual execution time of a task is usually unknown.

Instead, we consider a range of possible execution times bounded by the best and worst-case execution times of the task. Hence, if a higher-priority task is released when a lower-priority task is already in execution, we cannot give an exact point of execution where the lower-priority task is guaranteed to be at the time. Thus, there could arise a situation where the lower-priority task *could* be inside its non-preemptive region but is *not guaranteed* to be.

In our work, we consider a periodic real-time task model with period equal to the deadline of a task. The notation used in the remainder of this paper is as follows. A task T_i has characteristics represented by the 7 tuple (Φ_i, P_i, C_i, c_i, B_i, R_i, $\Delta_{j,i}$). Here, Φ_i is the phase, P_i is the period (equal to deadline), C_i is the worst-case execution time, c_i is the best-case execution time, B_i is the blocking time and R_i is the response time of the task. $\Delta_{j,i}$ is the preemption delay inflicted on the task due to a higher priority task T_j. $J_{i,j}$ represents the jth instance (job) of task T_i.

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 gives an overview of prior work on completely preemptive analysis. Section 4 discusses our methodology and Section 5 presents experimental results of our analysis. We summarize the contributions of our work in Section 6.

2. Related Work

Recently, there has been considerable research in the area of data cache analysis for real-time systems. Several methods characterize data cache behavior with respect to a single task. Recently, some analytical methods for characterizing data cache behavior were proposed [8, 6, 5]. In prior work [13], we extended the Cache Miss Equations framework by Ghosh *et al.* [8] to produce exact data cache reference patterns.

Several techniques have been proposed to analyze tasks and calculate preemption delay in multi-task, preemptive environments. Lee *et al.* proposed and enhanced a method to calculate an upper bound for cache-related preemption delay (CRPD) in a real-time system [10, 11]. They used cache states at basic block boundaries and data flow analysis on the control-flow graph of a task to analyze cache behavior and calculate preemption delays.

The work by Lee *et al.* was enhanced by Staschulat *et al.* [16, 18]. They build a complete framework for response time analysis of tasks. Their focus is on instruction caches rather than data caches. Consideration of data caches is fundamentally different from consideration of instruction caches because the actual memory addresses accessed by the same reference in multiple iterations may be different in the case of data accesses. Due to this fact, the methodology used to analyze instruction caches is not suitable for analyzing data caches. In prior work [14, 15], we propose a framework following similar steps to calculate worst-case response time as the work by Staschulat *et al.* but using a significantly different methodology.

More recently, Staschulat *et al.* proposed a framework to calculate WCET of tasks [17]. This framework considers both input-independent and input-dependent accesses and calculates a tight bound of the effect of input-dependent accesses on input-independent ones. When unpredictable data references exist, any reused data cache contents are assumed to be replaced, forcing them to assume that the entire data cache is replaced in case of arrays larger than the data cache size. In our work, we only focus on predictable (input-independent) data cache accesses. Furthermore, we need not make any assumptions about array sizes with respect to data cache size.

In other related work, Ju *et al.* propose a method to extend CRPD calculations using abstract cache states to dynamic scheduling policies [9]. Once again, this work focuses on instruction caches. Our handling of data caches differs significantly.

There have been several pieces of work that provide schedulability analysis and tests for non-preemptive systems [7]. However, their fundamental assumption is that every task is completely non-preemptive. They do not allow any task to be partially or fully preemptive. This assumption simplifies analysis greatly but decreases schedulability of task sets. In order to increase schedulability, yet achieve lower analysis complexity, methods were proposed to "defer" preemptions to known points in time by splitting a job into several small sub-jobs and allowing preemptions only at the end of a sub-job [3, 4, 12]. Recent work by Bril et al. demonstrates flaws in this method [2, 1].

3. Prior Work

In previous work, we presented a framework that statically analyzes tasks in a multi-task preemptive environment and produces safe and tight worst-case response time estimates for tasks [15]. When a task is preempted, some data that it had loaded into the data cache may potentially be evicted from cache by higher-priority tasks. Hence, on resumption of execution, it incurs additional delay to bring the evicted data back into the data cache.

In order to incorporate the effects of preemption of a task by a higher-priority task, we perform three steps: (1) calculate *n*, the maximum number of preemptions for a task; (2) identify the worst-case placement of these preemptions in the iteration space of the preempted task; and (3) calculate the delay incurred by the task due to a specific preemption.

Our analysis presented a framework that calculated a safe and tight estimate of the maximum number and the placement of preemptions for a task by eliminating infeasible preemption points. A preemption point is infeasible for a certain task if the task has not started at all before the point or if the task has already completed execution before

the point. We used both the best and the worst-case execution times of higher priority tasks to help tighten the actual preemption delay at every identified preemption point.

Our method showed significant improvements over a prior method proposed by Staschulat *et al.* [16, 18] and over theoretical bounds for the maximum number of preemptions. We also showed that, when preemption delay is accounted for, the critical instant for a task set does not necessarily occur when all tasks in the task set are released simultaneously as is generally assumed.

4. Methodology

Section 3 briefly discusses prior work in which we propose a method to calculate the worst-case response time of a task in a multi-task preemptive hard real-time system [15]. In that work, the basic assumption is that a task may be preempted **at any point** during its execution by a task with higher priority. Hence, we are unable to consider task sets with tasks that contain a non-preemptive region (NPR) within them. Our current work aims at proposing a methodology that allows such tasks.

In the work presented here, we assume that every task has at most one NPR during its execution. Conceptually, our framework can deal with tasks that have multiple NPRs during their execution, a feature to be incorporated in the implementation in the future. However, this feature increases the complexity of the analysis as a function of the number of non-preemptive regions. As part of future work, we intend to develop a more efficient method to handle tasks with multiple non-preemptive regions.

A NPR is represented by the first and last points of the range of consecutive iteration points during which a particular task may not be interrupted. Every task is hence effectively divided into three regions with the middle one representing the NPR. The static timing analyzer described in prior work [15] is enhanced to calculate the worst-case and best-case execution times of these three regions based on the start and end iteration points of the NPR.

In our prior work [15], whenever an instance of a task is released, it is placed in a service queue and the scheduler is invoked. The scheduler chooses the task with the highest priority at the current time, preempting any lower priority task that might be executing at the time. However, in our new system, a task with higher priority may be required to wait if a lower-priority task is executing in its NPR. In order to calculate the worst-case response time for every task, we need to consider several possible scenarios.

Let us suppose that a task T_1 is released at time t. At time $t + x$, a task T_0, with a higher priority than T_1 is released. At time $t + x$, there are three possible cases:

1. T_1 has finished executing its first region and started executing its NPR in both best and worst cases;
2. T_1 has not finished executing its first region in either case or has already finished its NPR and entered its third region in both cases; or
3. T_1 has started executing its NPR in the best-case, but not in the worst-case.

Cases 1 and 2 are straightforward. In case 1, T_0 has to wait until T_1 finishes executing its NPR. In the best case, this time is equal to the best-case remaining execution time of T_1's NPR. In the worst-case, it is equal to the worst-case remaining execution time of task T_1's NPR. In case 2, T_1 gets preempted and T_0 starts to execute immediately.

In case 3, it is not certain whether T_1 has started executing its NPR or not. Hence, for each task, we calculate the best and worst possible scenario for that particular task in order to determine its worst-case response time. For T_0, the worst case is to assume that T_1 has already started executing its NPR and add the worst-case remaining execution time of T_1's NPR to the response time of T_0. On the other hand, the best case for T_0 is to assume that T_1 has not yet started executing its NPR and, hence, may be preempted. The scenario is reversed for T_1. Its best case is to assume that it has already started executing its NPR and, hence, is not preempted. Its worst case is to assume that it gets preempted by T_0 and add the associated preemption delay to its remaining execution time. By considering parallel execution scenarios for each task, we can come up with safe response time estimates.

Currently, our framework assumes that, when a task is executing in its NPR, it cannot be preempted by any task. However, this is a matter of policy. The framework could easily be extended to support resource access protocols, such as the Priority Ceiling or Stack Resource Protocols, which strive to limit resource access conflicts. This change would be reflected in the handling of the JobRelease event shown in Figure 4.

4.1. Illustrative Examples

We now provide an illustrative example of our methodology. Consider the task set whose characteristics are specified in Table 1. The first column shows the task name. The second and third columns show the phase and period (equal to the deadline) of the each task. Let us assume that the Rate Monotonic (RM) scheduling policy is used for this task set and, hence, that the task with the shortest period has the highest priority. The fourth and fifth columns show the WCETs and BCETs of each of the three regions of each task.

For ease of understanding, let us also evaluate what happens if all three regions of every task are fully preemptive. Figure 1 shows the best and worst-case timelines below and above the horizontal time axis respectively. The arrows show release points of the three tasks. The lightly shaded rectangles represent preemptive execution regions of tasks.

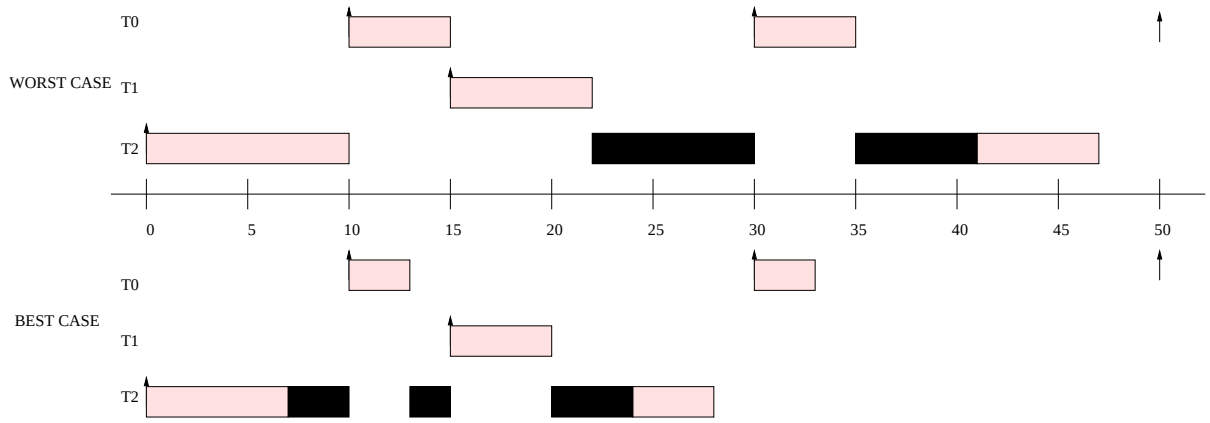


Figure 1. Best and Worst-Case Scenarios for Task Set 1 with no NPR

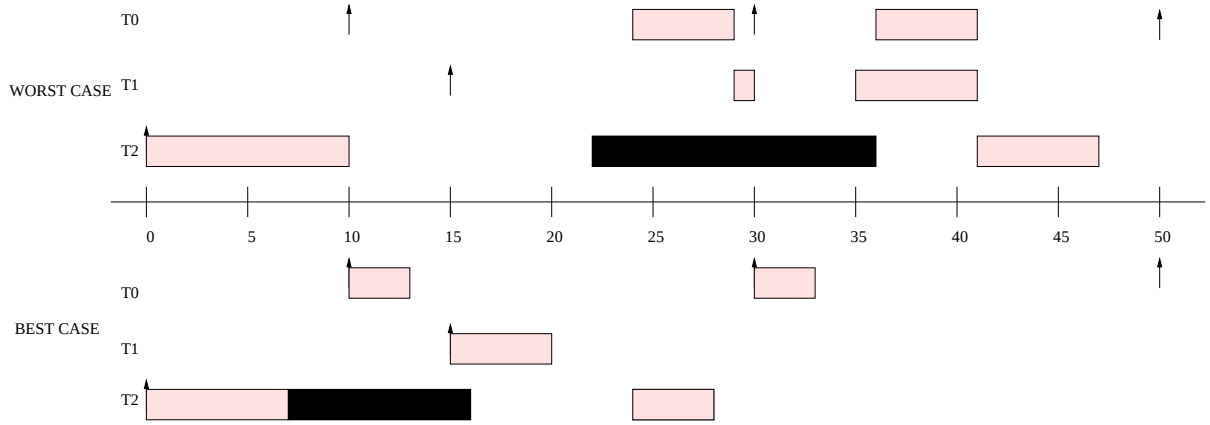


Figure 2. Best and Worst-Case Scenarios for Task Set 1 with NPR

Let us now add a non-preemptive region to task T_2, as indicated in Table 1. Figure 2 shows the timeline for this situation. Here, the black rectangles represent non-preemptive regions of execution. For the sake of comparison, the second region of task T_2 is shown as a black rectangle in Figure 1 although it is fully preemptive in that example.

In Figure 2, we observe that some execution regions overlap. This is because, at every release point, if there is some task that *could be* executing in its NPR but is *not guaranteed to be*, we consider best and worst case scenarios for that task and for the task released. In reality, only one of the scenarios takes place and there is no simultaneous execution of multiple tasks.

Task	**Phase**	**Period = deadline**	**WCET (r1/r2/r3)**	**BCET (r1/r2/r3)**
T_0	10	20	5/0/0	3/0/0
T_1	15	50	7/0/0	5/0/0
T_2	0	200	10/14/6	7/9/4

Table 1. Task Set Characteristics - Task Set 1 [RM policy $\rightarrow T_0$ has highest priority]

Due to space constraints, we shall not examine the entire timeline in detail. Instead, let us focus on three portions of the timeline shown. These portions will help explain the basic concept behind our methodology. Let us consider all the events that would occur at time 10. Job $J_{0,0}$ is released. In the case of a fully preemptive system (Figure 1), since $J_{0,0}$ has higher priority, it is scheduled immediately, preempting $J_{2,0}$. However, in the case where $J_{2,0}$ has a NPR (Figure 2), the situation is more complicated. Here, we need to consider two possibilities. The best case for $J_{0,0}$ is that it is scheduled immediately since there is a chance that $J_{2,0}$ has not yet started executing its NPR. It is scheduled to finish region 1 at time 13. On the other hand, since there is a chance that $J_{2,0}$ has started its NPR, $J_{0,0}$ has to wait for at most 14 units of time (worst-case remaining execution time of $J_{2,0}$'s NPR) and is scheduled to start only at time 24. The best case for $J_{2,0}$ is that it continues executing its NPR. However, in the worst case, since there is a chance that it has not started its NPR, it gets preempted by $J_{0,0}$ and it now re-scheduled to start its NPR at time 15 (adding the WCET of $J_{0,0}$). However, due to the release of another higher-priority job, namely, $J_{1,0}$, at time 15, $J_{2,0}$

gets re-scheduled once again to start at time 22 (adding the WCET of $J_{1,0}$).

Let us now move forward in the timeline to time 22. In Figure 1, this is the time at which $J_{2,0}$ starts executing its second region in the worst case. Similarly, in Figure 2, this is the time at which $J_{2,0}$ starts executing its NPR in the worst case. It is scheduled to finish this region at time 36. At time 24, $J_{0,0}$ starts executing region 1 in its worst case. It is scheduled to finish at time 29.

Now, consider the events that occur at time 30. Job $J_{0,1}$ is released. In the fully preemptive system (Figure 1), $J_{0,1}$ gets scheduled immediately since it has a higher priority, preempting $J_{2,0}$ in the worst case. In case of Figure 2, we see that, in the best case, $J_{0,1}$ starts executing region 1 right away and is scheduled to finish at time 33. However, in the worst case, since $J_{2,0}$ is guaranteed to have started its NPR, $J_{0,1}$ has to wait until $J_{2,0}$ completes its NPR and is, hence, scheduled at time 36.

The worst case for $J_{1,0}$ occurs when execution starts at time 29. $J_{0,1}$ is released at 30 and preempts $J_{1,0}$ once again for the duration of the WCET of $J_{0,1}$. Note that $J_{1,0}$ need not wait for the completion of the NPR of $J_{2,0}$. (It has already done so in the worst case.) Every job needs to wait at most once for a lower-priority task in its NPR.

The analysis proceeds in a similar fashion up to the hyperperiod of the task set, namely 200. In this example, for the sake of simplicity, preemption delay calculations are not shown. Delay at every resumption point is assumed to be zero. These calculations are in Section 4.3.

4.2. Analysis Algorithm

An algorithm briefly describing our methodology is shown in Figure 4. Our system is built on an event hierarchy. Every event has a handler which performs all operations necessary on the occurrence of the particular event. We have several event types, each with a priority, time of occurrence and information about the task and job that the event corresponds to. The events are ordered by time, and upon ties, by priority based on the type of event. The various events in our system, in order of priority, are BCEndExec, WCEndExec, DeadlineCheck, JobRelease, BCStartExec, WCStartExec and PreemptionDelayPhaseEnd. The algorithm in Figure 4 describes the actions that take place when a certain event is triggered. In the algorithm, we describe the events in an order that follows the flow of the logic rather than based on priority.

The basic flow of operations in our analysis is as follows. Stand-alone WCETs and BCETs are calculated for each region of each task. JobRelease and Deadline check events are pre-created based on task periods and inserted into a global event list. Events in the event list are handled one at a time until there are no more events. The basic life-

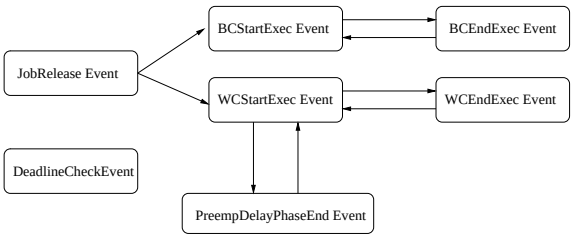


Figure 3. Creation Dependencies among Event Types

cycle of a job is described below. Upon release of a job, we evaluate when that job gets scheduled if possible and determine whether any job that is currently executing gets preempted due to this release. This triggers a B/WCStartExec event which signify the start of execution of the current region of a job. These events in turn schedule B/WCEndExec events or PreemptionDelayPhaseEnd events as the case may be. Finally, a DeadlineCheck is triggered and is responsible for checking if a certain job missed its deadline.

The creation of dependencies between event types are represented by the state-transition diagram shown in Figure 3. An arrow from one event type to another indicates that the handler of the first event type may create an event of the second type. Events that do not have a creator in the diagram are created at the beginning outside any of the event types.

4.3. Preemption Delay Calculation

Preemption delay at every identified preemption point is calculated in a manner consistent with our earlier work [15]. At every preemption point, we calculate the best-case and worst-case execution times that have been available for a task for its execution. We provide these values to the static timing analyzer and obtain the earliest and latest iteration points reachable for each of these times. We then consider the highest delay in this range of iteration points as given by the access chain weights for those points. In our past work, we simply added this delay to the remaining worst-case execution time of the task and assumed that, on resumption, execution continues from the iteration where it had left off. However, this is imprecise since we do not know at what points the preemption delay is actually incurred during the execution of the task. Hence, for future preemption points, determination of the iteration range where the task is supposed to be when it is preempted is not guaranteed.

In order to solve the above problem and provide safe estimates of the worst-case preemption delay at every point, we devised the following solution. When a task is preempted, we calculate the delay as indicated above. When the task later resumes execution, it enters a *preemption delay* phase for a time equal to the calculated delay. In this phase, the task prefetches all data cache items that contribute to the de-

```
bc_service_queue, wc_service_queue : queues of
   released jobs that have not yet completed
event_list : list of ordered events
current_time, curr_job : time and job of current event
JobRelease event: Release of new task instance.
if (event_queue is empty) {
   schedule StartExec event for curr_job at current_time
} else {
   if (curr_job has highest priority) {
      Best case: schedule StartExec event for curr_job
      Worst case: check currently executing job's NPR status
         schedule curr_job's StartExec event accordingly
         reschedule start of executing job if necessary
   } else {
      insert curr_job into event_queue according to priority
   } }
insert curr_job into b/wc_service_queue
B/WCStartExec event : Start execution of current region.
set status of curr_job to IN_SERVICE in b/wc_service_queue
if (curr_job is in PreemptionDelay phase) {
   schedule PreemptionDelayPhaseEnd event for curr_job
} else {
   schedule B/WCEndExec event for curr_job
}
B/WCEndExec event : End execution of current region.
remove curr_job from b/wc_service_queue
update b/wc_remaining_time of current job
if (curr_job has another region) {
   schedule B/WCStartExec event for next region if possible
   insert curr_job into b/wc_service_queue
} else if (b/wc_service_queue has more jobs in it) ) {
   schedule B/WCStartExec event of next READY job
}
DeadlineCheck event : Perform deadline check.
if (curr_job misses deadline) release its structures
PreemptDelayPhaseEnd event : End preemption delay phase
schedule WCStartExec event for curr_job
Main Algorithm : Starting point of analysis.
for every task in the task set {
     create JobRelease and DeadlineCheck events for all jobs
}
while (events in event list) {
   get highest priority event and handle it based on event type
}
```

Figure 4. Algorithm for NPR-Aware Calculation of WCET w/ Delay

lay. Once done, the task resumes normal execution. If a task gets preempted during its preemption delay phase, it pessimistically starts the same preemption delay phase all over again once it resumes execution. This new phase ensures that all future delay calculations are accurate.

As we can see from above, calculation of a tight bound of the preemption delay requires us to identify the range of iteration points where a task may be executing when it is preempted. In order to identify this range, relative phasing of jobs is required. Furthermore, no assumption can be made for the phasing of tasks that would result in the critical instant for the task set. Due to these reasons, we perform our analysis on a per-job basis rather than a per-task basis. Since our analysis is a static, offline one, we believe the complexity is acceptable. However, by using a mathematical formulation for our analysis (the derivation of which is part of ongoing work) and by assuming maximum possible preemption delay at every identified preemption point (thus eliminating the need to identify ranges of iteration points), it is possible to reduce the complexity, yet yield a more pessimistic bound.

5. Experimental Results

For our experiments, we constructed several task sets using benchmarks from the DSPStone benchmark suite [19], consistent with earlier work [15]. These task sets have base utilizations of 0.5, 0.6, 0.7 and 0.8. For each of these utilizations, we construct task sets with 2, 4, 6 and 8 tasks. For a utilization of 0.8, we also construct a task set with 10 tasks. In all our experiments, we use a 4KB, direct-mapped data cache with a hit penalty of 1 cycle and a miss penalty of 100 cycles. The stand-alone WCETs and BCETs of the various benchmarks are depicted in Table 2. The prefixed numbers in some of the benchmarks indicate the number of iterations. In all benchmarks, with the exceptions of matrix1 and dot-product, the number of iterations is 100 in cases where there is no prefix.

ID	Name	WCET	BCET	ID	Name	WCET	BCET
1	convolution	7491	7491	15	matrix1	59896	54015
2	200convolution	14191	14191	16	fir	9537	9537
3	300convolution	20891	20891	17	500fir	43937	43937
4	500convolution	34291	34291	18	600fir	54837	52537
5	600convolution	45291	40991	19	700fir	65937	61137
6	700convolution	55491	47691	20	800fir	77037	69737
7	800convolution	66191	54391	21	900fir	88137	78337
8	900convolution	76391	61091	22	1000fir	99237	86937
9	1000convolution	87091	67791	23	lms	14536	14536
10	n-real-updates	16738	16738	24	600lms	89636	79536
11	300n-real-updates	56538	47338	25	700lms	112636	92536
12	400n-real-updates	92238	62638	26	800lms	135636	105536
13	500n-real-updates	127538	77938	27	900lms	158636	118536
14	dot-product	750	750	28	1000lms	181636	131536

Table 2. Stand-Alone WCETs and BCETs of DSPStone Benchmarks

In our first set of experiments, we perform response time analysis using the method presented in this paper to calcu-

ID	Region 1	Region 2 (NPR)	Region 3
	WCET / BCET	WCET / BCET	WCET / BCET
5	39371 / 38271	5084 / 2184	836 / 536
6	39371 / 38971	10924 / 6024	5196 / 2696
7	46771 / 44471	14224 / 7224	5196 / 2696
8	52371 / 48771	18824 / 9624	5196 / 2696
9	61571 / 55471	15424 / 7224	10096 / 5096
11	33494 / 31194	5337 / 3737	17707 / 12407
12	52294 / 43194	9647 / 4847	30297 / 14597
13	68444 / 53344	12987 / 5587	46107 / 19007
15	28912 / 26172	22400 / 20760	8584 / 7083
17	32302 / 32302	9045 / 9045	2590 / 2590
18	45802 / 45402	5845 / 4545	3190 / 2590
19	58652 / 55352	5845 / 4545	1440 / 1240
20	56502 / 53602	11545 / 9045	8990 / 7090
21	69352 / 63552	11545 / 9045	7240 / 5740
22	70152 / 64052	17245 / 13545	11840 / 9340
24	47756 / 45956	4649 / 3549	37231 / 30031
26	66506 / 60406	5239 / 3939	63891 / 41191
27	59256 / 54956	20639 / 15639	78741 / 47941

Table 3. Characteristics of Regions of Tasks with NPR

late the number of preemptions and the worst-case preemption delay. Due to the fact that the benchmarks used in our experiments do not already have a NPR, we simply choose an iteration range from the valid iteration range of a particular task and mark it as being non-preemptive. Table 3 shows execution times of each region as determined by the timing analyzer based on the chosen iteration ranges for a subset of our benchmarks. Since we only have a fixed set of benchmarks, we sometimes use the same benchmark with and without NPRs in different task sets. The length of a task's NPR as a portion of its total execution time ranges from 4% to 37% in both the worst and the best cases.

The characteristics of task sets with base utilization 0.5 and 0.8 are shown in Table 4. The characteristics and results for utilizations 0.6 and 0.7 are omitted due to space constraints. The 1st column shows the tasks used in each task set. We use the IDs assigned to benchmarks in Table 2 to identify the tasks. If a task is chosen to have a NPR in a certain task set, we append the letter *N* to its ID to indicate this fact. In this case, the WCETs and BCETs for the task are as shown in Table 3. Otherwise, they are as indicated in Table 2. The 2nd column shows the phases of the tasks and the 3rd column shows the periods (equal to the deadlines) of tasks. The phases of the tasks are chosen in a way to demonstrate interesting features of our analysis.

Results obtained for task sets in the above set of experiments are shown in Figures 5 and 6 for base utilizations of 0.5 and 0.8, respectively. Each graph shows the results of analysis of the same task sets using both the static Rate Monotonic (RM) scheduling policy and the dynamic Earli-

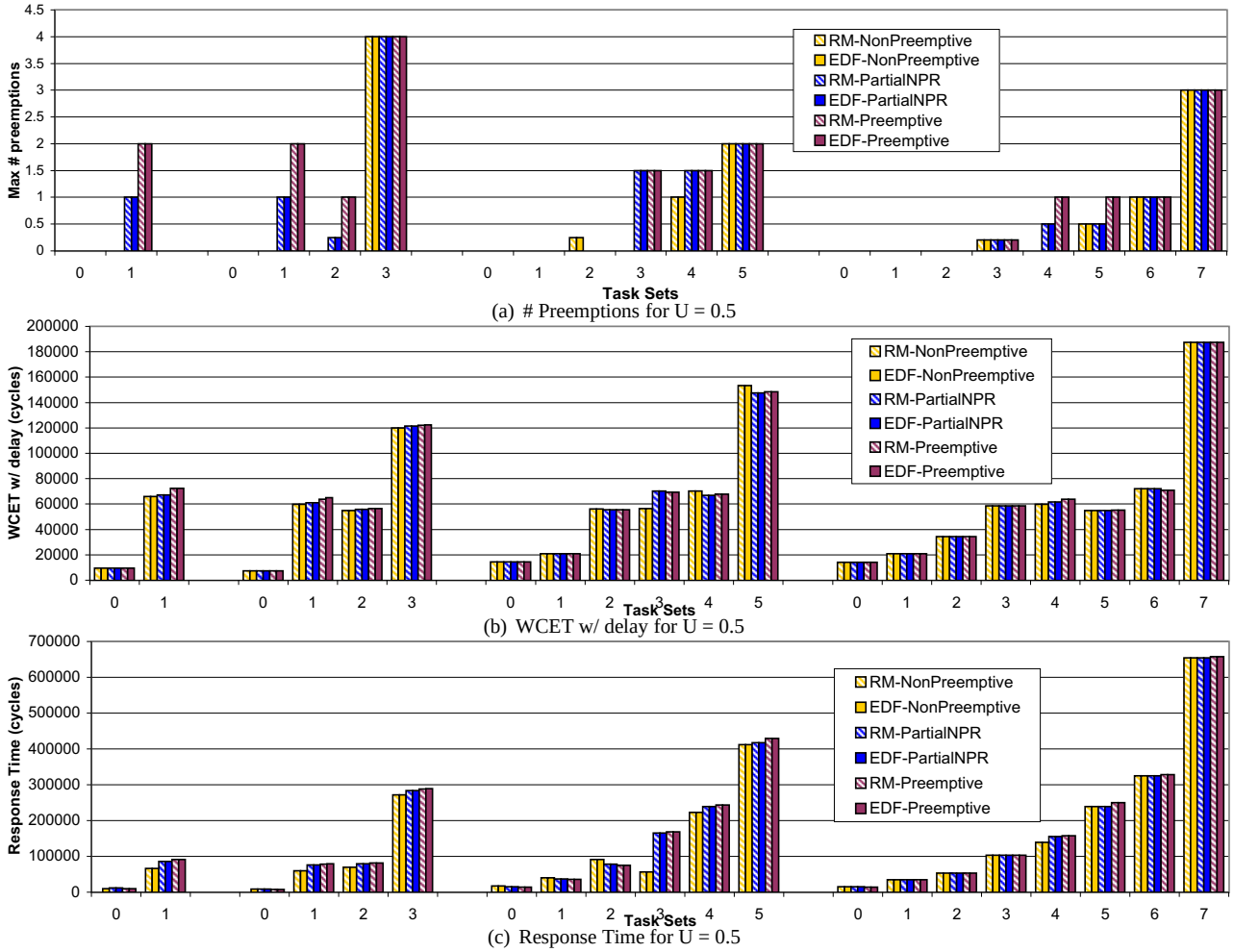
# Tasks	2	4	6	8
U = 0.5				
IDs	16, 19N	1, 15N, 18N, 22	23, 3, 6, 11N, 19, 26	2, 3, 4, 11, 15N, 18, 7, 27
Phases	4K, 0	1K, 0, 10K, 0	32K, 32K, 32K, 0, 0, 0	0, 0, 0, 0, 0, 0, 0, 0
Periods	50K, 200K	50K, 400K, 500K, 1000K	400K, 500K, 1000K, 1000K, 2000K	100K, 400K, 500K, 800K, 1000K, 2000K, 2000K, 4000K
U = 0.8				
IDs	27, 26N	28, 13N, 27, 19	21, 8N, 20, 13, 25, 19	8, 26, 20, 15N, 9, 11, 8, 21
Phases	0, 0	54K, 0, 0, 0	49K, 0, 0, 0, 0, 0	27K, 27K, 27K, 0, 0, 0, 0, 0
Periods	300K, 500K	500K, 500K, 1000K, 2000K	400K, 500K, 500K, 1000K, 1000K, 2000K	400K, 500K, 800K, 800K, 1000K, 2000K, 2000K, 4000K
U = 0.8, # Tasks=10				
IDs	10, 8, 15, 9, 5, 11N, 20, 27, 22, 17			
Phases	32K, 32K, 32K, 32K, 32K, 0, 0, 0, 0, 0			
Periods	100K, 625K, 625K, 625K, 1000K, 1000K, 1250K, 1250K, 2500K, 5000K			

Table 4. Task Set Characteristics: Benchmark IDs, Phases[cycles] and Periods [cycles]

est Deadline First (EDF) scheduling policy. For each utilization, we have a separate graph for the maximum number of preemptions, the WCET with preemption delay and the response time. These values form the y-axes in the graphs. In each case, we indicate the average values of these parameters over all jobs of a task. On the x-axis for each graph, we show the tasks used in each experiment. Tasks are grouped by task-set and by task-id starting from 0 within task sets.

For each scheduling policy, we show results using three analysis techniques. The first one is NPR unaware (Preemptive), in which all tasks are assumed to be completely preemptive, as in our earlier work [15]. The second is a NPR-aware analysis, in which some tasks have a non-preemptive region in the middle (PartialNPR). The third analysis is a NPR-aware analysis, in which the tasks with a non-preemptive region are assumed to be completely non-preemptive (NonPreemptive). The results for fully non-preemptive schedules are obtained using the algorithm described in Figure 4 and by setting the lengths of the first and third regions to zero. In these graphs, we omit response time values for tasks that end up missing their deadline.

At the outset, it is to be noted that, if a task is supposed to have a non-preemptive region, then forcing the task to



(a) # Preemptions for U = 0.5

(b) WCET w/ delay for U = 0.5

(c) Response Time for U = 0.5

Figure 5. Results for U=0.5 under RM and EDF Scheduling

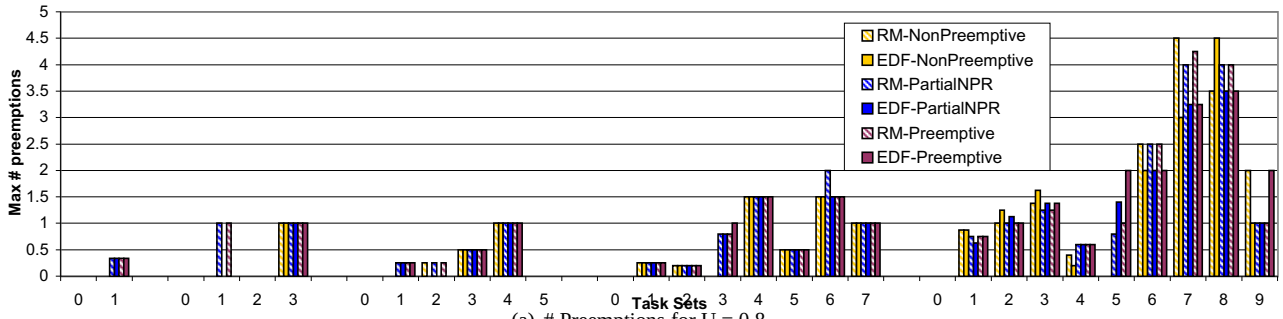
be completely preemptive is unsafe since the results of the task could be incorrect (due to possible data races). Hence, the results of our NPR unaware (Preemptive) analysis are unsafe as far as the tasks with NPR are concerned. It is purely for the sake of comparison that we present those results here. On the other hand, making a task that is supposed to have a portion which is non-preemptive completely non-preemptive is conservative, yet safe.

From the graphs, we make the following observations. First of all, we observe that the results for the RM scheduling policy and the EDF scheduling policy are almost the same for most tasks. For RM and EDF to exhibit a difference in behavior, a task with a longer period needs to have an earlier deadline than one with shorter period somewhere in the execution timeline. This could happen in two situations, namely, when the shorter period does not divide the longer period and when there is phasing between the tasks. In most of our task sets, neither case occurs as observable from the results. However, for a base utilization of 0.8, we do observe small differences in the two policies. As expected, some tasks with a shorter period (higher priority according to RM) have a longer response time with EDF. Other tasks in the same task set with a longer period have a shorter response time with EDF.

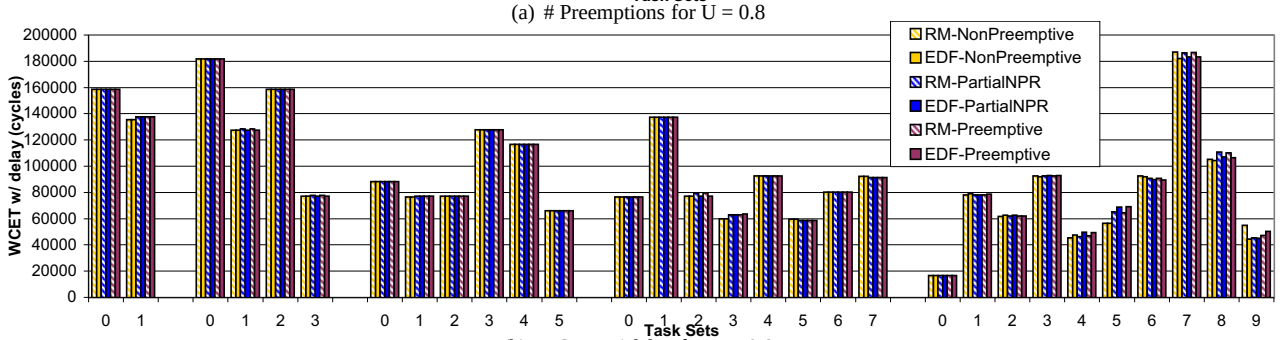
For most of our task sets, we observe that the response time estimates obtained from the NonPreemptive analysis is shorter than that obtained from the PartialNPR analysis. The reason for this is as follows. In the PartialNPR analysis, the following situation could occur. When a task is released, some task with a lower priority could have started its NPR in the best case, but not started it in the worst.

As explained in Section 4, when this happens, we consider the effects of contradicting worst-case scenarios for the two tasks involved. In other words, we assume the worst possible scenario for each task. This is done in order to ensure safety of the response time estimates. In reality, however, only one of the scenarios can actually occur. In the case of the NonPreemptive analysis, a task that has a NPR is assumed to be completely non-preemptive. Hence, a situation like the one described above cannot occur.

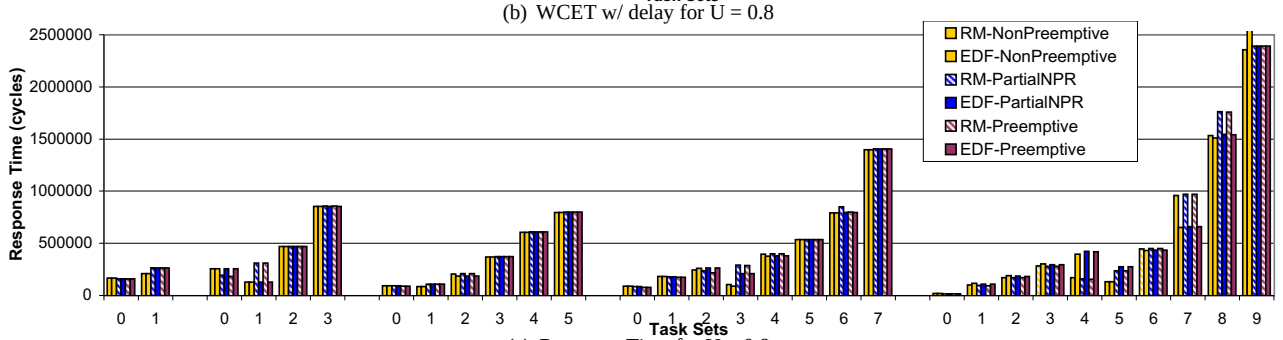
On the other hand, in some task sets, the NonPreemptive analysis causes some high-priority tasks to miss their dead-



(a) # Preemptions for U = 0.8



(b) WCET w/ delay for U = 0.8



(c) Response Time for U = 0.8

Figure 6. Results for U=0.8 under RM and EDF Scheduling

lines. This is because the waiting time for the high-priority tasks are now longer since the length of the non-preemptive region of a task extends to its entire execution time. This, in part, compensates for the pessimism that the PartialNPR method introduced and is observed by the fact that the actual difference between response times of tasks in the two cases are not significant.

We also conducted a sensitivity study using the example task set shown in Table 1. We maintain the same periods, phases and total execution times for all tasks. However, we vary the length of the NPR in T_2 in both the best and worst cases. We start without a NPR for T_2 and then extend the NPR from the middle outwards symmetrically in both directions until T_2 is completely non-preemptive. Table 5 shows the WCETs and BCETs of each region for different experiments. The average response times over all jobs of each task using the RM scheduling policy are shown in Figure 7. Response times are omitted from the graph if any job of a task misses its deadline. At one extreme, where T_2 is completely preemptive, we see that the response time of T_0 is the same as its WCET since it executes to completion right after its initial release. At the other extreme, when T_2 is completely non-preemptive, we see that T_0 misses its deadline due to increased waiting time. This sensitivity study demonstrates the improved schedulability of our PartialNPR analysis over the NonPreemptive analysis.

Expt. #	1	2	3	4	5	6	7	8
Region1	30/20	13/9	11/8	9/7	7/6	5/4	3/2	0/0
Region2:NPR	0/0	4/2	8/4	12/6	16/8	20/12	24/16	30/20
Region3	0/0	13/9	11/8	9/7	7/6	5/4	3/2	0/0

Table 5. WCET/BCET ratios for T_2

In summary, our work enables us to study the effects of having a non-preemptive region and the advantages of having partial NPRs as compared to completely non-preemptive tasks in a task set. Assuming that a task is completely non-preemptive, though simpler to analyze, has the disadvantage that there is an increased probability of some high-priority task missing its deadline. On the other hand, a completely preemptive system might not be acceptable for certain kinds of tasks that inherently pos-

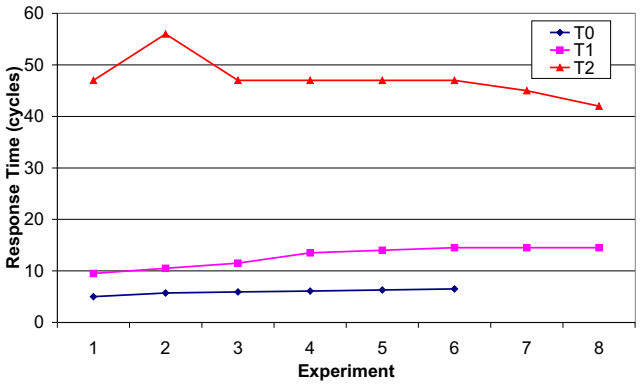


Figure 7. Response Times of Tasks

sess a region in which they should not be preempted in order to preserve correctness. In such cases, our analysis may be used to calculate whether the task set is schedulable or not.

6. Conclusion

We presented a framework to calculate safe and tight timing bounds of data-cache related preemption delay (D-CRPD) and worst-case response times. In contrast to past work, our novel approach handles tasks with a non-preemptive region of execution. Through experiments, we obtain response-time bounds for task sets where some tasks have non-preemptive regions. We compare these results to an equivalent task set where only those tasks with non-preemptive regions are scheduled non-preemptively altogether. We show that, for some task sets, schedulability is improved without significantly affecting the response times of tasks using partially non-preemptive tasks as opposed to fully non-preemptive tasks. To the best of our knowledge, this is the first framework that bounds D-CRPD and response times for tasks with non-preemptive regions.

References

[1] R. Bril. Existing worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption refuted. In *Work in Progress (WiP) session of the 18th Euromicro Conference on Real-Time Systems*, July 2006.

[2] R. Bril, J. Lukkien, and W. Verhaegh. Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption revisited. Cs-report 06-34, Technische Universiteit Eindhoven (TU/e), The Netherlands, Dec. 2006.

[3] A. Burns. Pre-emptive priority based scheduling: An appropriate engineering approach. *S. Son, editor, Advances in Real-Time Systems*, pages 225–248, 1994.

[4] A. Burns and A. Wellings. Restricted tasking models. In *8th International Real-Time Ada Workshop*, pages 27–32, 1997.

[5] S. Chatterjee, E. Parker, P. Hanlon, and A. Lebeck. Exact analysis of the cache behavior of nested loops. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 286–297, June 2001.

[6] B. B. Fraguela, R. Doallo, and E. L. Zapata. Automatic analytical modeling for the estimation of cache misses. In *International Conference on Parallel Architectures and Compilation Techniques*, 1999.

[7] L. George, N. Rivierre, and M. Spuri. Pre-emptive and non-pre-emptive real-time uni-processor scheduling. Technical report, Institut National de Recherche et Informatique et en Automatique (INRIA), France, Sept. 1996.

[8] S. Ghosh, M. Martonosi, and S. Malik. Cache miss equations: a compiler framework for analyzing and tuning memory behavior. *ACM Transactions on Programming Languages and Systems*, 21(4):703–746, 1999.

[9] L. Ju, S. Chakraborty, and A. Roychoudhury. Accounting for cache-related preemption delay in dynamic priority schedulability analysis. In *IEEE Design Automation and Test in Europe*, 2007.

[10] C.-G. Lee, J. Hahn, Y.-M. Seo, S. L. Min, R. Ha, S. Hong, C. Y. Park, M. Lee, and C. S. Kim. Analysis or cache-related preemption delay in fixed-priority preemptive scheduling. *IEEE Transactions on Computers*, 47(6):700–713, 1998.

[11] C.-G. Lee, K. Lee, J. Hahn, Y.-M. Seo, S. L. Min, R. Ha, S. Hong, C. Y. Park, M. Lee, and C. S. Kim. Bounding cache-related preemption delay for real-time systems. *IEEE Transactions on Software Engineering*, 27(9):805–826, Nov. 2001.

[12] S. Lee, C.-G. Lee, M. Lee, S. Min, and C.-S. Kim. Limited preemptible scheduling to embrace cache memory in real-time systems. In *ACM SIGPLAN Workshop on Languages, Compilers and Tools for Embedded Systems (LCTES), Lecture Notes in Computer Science (LNCS) 1474*, pages 51–64, 1998.

[13] H. Ramaprasad and F. Mueller. Bounding worst-case data cache behavior by analytically deriving cache reference patterns. In *IEEE Real-Time Embedded Technology and Applications Symposium*, pages 148–157, Mar. 2005.

[14] H. Ramaprasad and F. Mueller. Bounding preemption delay within data cache reference patterns for real-time tasks. In *IEEE Real-Time Embedded Technology and Applications Symposium*, Apr. 2006.

[15] H. Ramaprasad and F. Mueller. Tightening the bounds on feasible preemption points. In *IEEE Real-Time Systems Symposium*, pages 212–222, Dec. 2006.

[16] J. Staschulat and R. Ernst. Multiple process execution in cache related preemption delay analysis. In *ACM International Conference on Embedded Software*, 2004.

[17] J. Staschulat and R. Ernst. Worst case timing analysis of input dependent data cache behavior. In *Euromicro Conference on Real-Time Systems*, 2006.

[18] J. Staschulat, S. Schliecker, and R. Ernst. Scheduling analysis of real-time systems with precise modeling of cache related preemption delay. In *Euromicro Conference on Real-Time Systems*, 2005.

[19] V. Zivojnovic, J. Velarde, C. Schlager, and H. Meyr. Dspstone: A dsp-oriented benchmarking methodology. In *Signal Processing Applications and Technology*, 1994.