Exact Schedulability Analysis for Static-Priority Global Multiprocessor Scheduling Using Model-Checking[⋆]

Nan Guan[1], Zonghua Gu[2], Qingxu Deng[1], Shuaihong Gao[1], and Ge Yu[1]

[1] Department of Computer Science and Engineering
Northeastern University, Shenyang, China
[2] Department of Computer Science and Engineering
Hong Kong University of Science and Technology, Hong Kong, China

Abstract. To determine schedulability of priority-driven periodic tasksets on multi-processor systems, it is necessary to rely on utilization bound tests that are safe but pessimistic, since there is no known method for exact schedulability analysis for multi-processor systems analogous to the response time analysis algorithm for single-processor systems. In this paper, we use model-checking to provide a technique for exact multi-processor scheduability analysis by modeling the real-time multi-tasking system with Timed Automata (TA), and transforming the schedulability analysis problem into the reachability checking problem of the TA model.

1 Introduction

For single-processor systems, there are mainly two approaches to schedulability analysis: *utilization bound tests* and *response time analysis*. Take fixed-priority Rate Monotonic (RM) scheduling for example. The well-known Liu and Layland utilization bound test [1] states that a taskset with N tasks is schedulable if the total utilization does not exceed $N(2^{1/N}-1)$. This is a sufficient but not necessary condition, and rejects some tasksets that are schedulable. In fact, all utilization bound tests are necessarily pessimistic. Lehoczky *et al* [2] presented response time analysis, a polynomial-time algorithm for calculating a task's Worst-Case Response Time (WCRT) by performing processor demand analysis when the task and all other higher-priority tasks are initially released at time 0, the *critical instant*. A task is schedulable if its WCRT is less than its deadline, and the taskset is schedulable if all tasks are schedulable. This is a necessary and sufficient condition for schedulability.

Multiprocessor (MP) systems are drawing a lot of attention recently, with industry trends such as multi-core processors and Multiprocessor Systems-on-a-Chip (MPSoC), hence real-time scheduling and schedulability analysis for MP systems become an important research area. MP scheduling algorithms can be

[⋆] This work is partially supported by National Basic Research Program of China (973 Program) under Grant No.2006CB303000 and the Cultivation Fund of the Key Scientific and Technical Innovation Project, Ministry of Education of China, under Grant No.706016.

classified into three categories, (no migration, restricted migration and full migration) based upon the permissible degree of inter-processor migration [3].

No migration (partitioned) scheduling with a given task allocation to processors is similar to single-processor scheduling and can be addressed with existing techniques, but restricted and full migration scheduling brings serious challenges to schedulability analysis. For these task models, an analogous algorithm for WCRT calculation does not exist, since there may not be a *critical instant* as in single-processor scheduling. Traditionally, there are two methods to determine the schedulability of MP systems: *utilization bound tests*, which is safe but pessimistic, and *simulation*, which is unsafe, since it only explores one execution trace, not exhaustive exploration of the state space. For simulation, a widely adopted convention is to set all task release offsets to be zero. However, in contract to single-processor scheduling, it is not necessarily true that this is the worst case situation that maximizes task response times for MP scheduling, hence simulation sometimes gives the wrong result, i.e., determine a taskset to be schedulable even though it is not.

In view of the drawbacks of utilization bound tests and simulation, it would be valuable if we could have a method for exact schedulability analysis without the pessimism of the utilization bound tests. In this paper, we provide an exact method for static-priority MP schedulability analysis without any pessimism by transforming the schedulability problem into reachability analysis problem of Timed Automata. In addition to exact schedulability analysis of periodic tasksets, model-checking has an additional benefit of being able to handle non-periodic tasksets. In classic scheduling theory, real-time tasks are usually assumed to be periodic, and sporadic tasksets are treated as periodic ones using the minimum inter-arrival time as the task period, which results in pessimistic analysis results. But with model-checking, we can model the external environment that triggers the taskset in a precise manner, thus avoiding the pessimism of the strict periodic taskset assumption.

We make a number of simplifying assumptions in this paper. We assume that tasks are assigned static priorities, and they are independent from each other without precedence relationships and data sharing. Each task has a fixed execution time instead of a range of possible execution times. Although it is not difficult to relax these assumptions in our modeling framework, we make these assumptions for the sake of clarity of presentation.

This paper is organized as follows. We first discuss related work in Section 2. We present the TA model for restricted migration in Section 3 and for full migration scheduling in 4. We present performance evaluation results in Section 5. Finally, we draw conclusions in Section 6.

2 Related Work

2.1 Utilization Bound Tests

For EDF-based MP scheduling, several authors have presented utilization bound tests. Goossens et al [5] presented a test assuming that tasks have relative deadlines equal to the period. Baker [6] presented another test that can handle relative deadlines less than or equal to the period. Baker [7] extended [6] to include tasks with post-period deadlines, and showing that EDF-US[1/2], which gives higher priority to tasks with utilizations above 1/2, is optimal. Bertogna et al [8] presented an improved test, and showed that it is incomparable to [6], and each test can accept tasksets that the other test rejects. For tasksets with different timing characteristics, they have different performance in terms of acceptance ratio.

For fixed-priority MP scheduling, Andersson [9] proved that the utilization guarantee for any static-priority MP scheduling algorithm, cannot be higher than $(m+1)/2$ for an m-processor platform. This conclusion places a theoretical upper bound of the utilization bound test for MP scheduling, and highlights the inherent pessimistic natural of the schedulability bound tests. For full-migration static priority scheduling, Andersson [9] defined a periodic taskset with constrained deadlines [1] to be a light system on m processors if it satisfies the following properties: (1)$\sum_{i=1}^{N} \frac{C_i}{T_i} \leq \frac{m^2}{3m-2}$, (2) $\frac{C_i}{T_i} \leq \frac{m}{3m-2}$, for $1 \leq i \leq N$, and showed that any periodic task system that is light on m processors is schedulable on m processors with preemptive RM algorithm.

Baruah [17] proved a similar result with the conclusion that a taskset, with all deadlines equal to periods, is guaranteed to be schedulable on m processors with RM scheduling if $C_i/T_i \leq 1/3$ for $1 \leq i \leq N$ and $C_i/T_i \leq m/3$. The group of tests consist of three tests with complexity of $O(N^3)$, $O(N^2)$ and $O(N)$ respectively.

Baker [16] presented a group of efficiently computable schedulability tests for fixed-priority scheduling of periodic tasksets with arbitrary deadlines on a homogeneous MP system. They improve upon Andersson's utilization bound tests by relaxing the assumptions of rate monotonic priorities and deadline being equal to period. For the special case when deadline equals period and priorities are rate monotonic, any set of tasks with maximum individual task utilization u_{max} and minimum individual task utilization u_{min} is feasible if the total utilization does not exceed $m(1 - u_{max})/2 + u_{min}$. We will compare our approach to the utilization bound tests in Andersson [9] and Baker [16] in Section 5.

2.2 Formal Methods for Schedulability Analysis

TIMES [4] is a tool for schedulability analysis of periodic or sporadic tasksets on a single processor. It uses Extended Timed Automata with asynchronous processes to model the real-time taskset, and UPPAAL [12] as the analysis engine. Fersman [10] showed that, for fixed-priority scheduling on a single processor, the schedulability checking problem can be transformed into reachability analysis on TA using only two extra clocks in addition to the clocks to describe task arrival times. This observation greatly reduces the state space and improves scalability of model-checking, since the state space increases sharply with the number of real-time clocks. However, TIMES is only applicable to single-processor systems, while we extend its approach to handle MP scheduling in this paper.

[1] The deadline of a periodic task is constrained if its relative deadline is equal to its period.

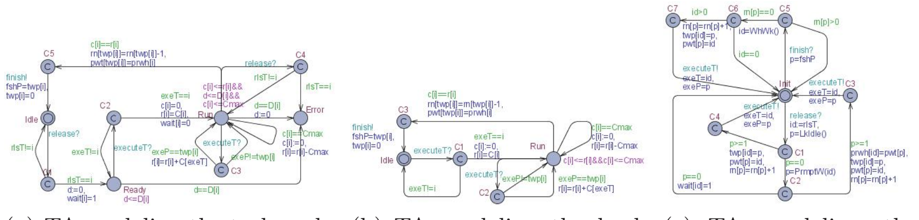
ACSR-VP [11] stands for Algebra of Communicating Shared Resources with Value Passing, a real-time process algebra used to model and solve the schedulability analysis as well as priority assignment problems. UPPAAL does not have parametric analysis capability of ACSR, so it can be used for schedulability analysis but not for priority assignment. Conceptually, we could have used ACSR-VP to model and solve the MP schedulability problem instead of UPPAAL. It is not our purpose to compare strengths and weaknesses of different modeling formalisms and tools, so we leave this as possible future work.

3 TA Model for Restricted Migration Scheduling

We have two alternatives approaches for building the TA model. The first one is to model all the tasks within a single model. This approach requires two clocks in each task automaton, one for accumulation of execution time in order to know when a job finishes execution, and the other one for testing if a task has missed its deadline. With this approach, $2N$ clocks are involved if there are N tasks.

We take advantage of these restrictions to reduce the number of clocks. Since a high-priority task will never be delayed by low-priority tasks, we model and check schedulability of each task one by one in decreasing order of priority, similar to the approach of TIMES. When we are checking schedulability for a task T_i, T_i is called the *task under analysis*, and all other tasks with higher priority than T_i are called the *background tasks*. The tasks with lower priority than T_i do not need to be modeled. We need to use one clock in the TA modeling each background task to accumulate its execution time, and use two clocks in the TA modeling the task under analysis. Therefore, the maximal number of clocks is $N + 1$, and model-checking needs to be performed for at most N times. Since the state space of timed automata grows drastically with the number of clocks, this alternative is superior to the first one.

As discussed earlier, model-checking is done for each task in decreasing order of priority, and only the tasks with higher priority than task i are modeled in S when task i is the task under analysis. The automaton S is the parallel composition of one task automaton $v - task$ modeling the task under analysis(Fig. 1(a)), i



(a) TA modeling the task under analysis (b) TA modeling the background task (c) TA modeling the scheduler

Fig. 1. TA model for restricted migration scheduling

task automata $non-v-task$ modeling each background task (Fig. 1(b)), where i is the number of the tasks with higher priority than the task under analysis, and one automaton *scheduler* modeling the scheduler (Fig. 1(c)). In Fig. 1(a), the task automaton is initially in location $Idle$. When the *scheduler* automaton issues an event $executeT!$, each task automaton checks to see if this command is meant for itself. If yes ($exeT == i$), then it goes into location Run; otherwise ($exeT! = i$), it goes back to location $Idle$. When task i is in location Run, some other jobs with higher priority may be allocated to its processor and preempt it ($Run \rightarrow C2 \rightarrow Run$). As shown in Fig. 2, when task i is preempted, $r[i]$ is updated, and task i finishes execution when $c[i] == r[i]$. At this time, clock $c[i]$ is reset, and the variable $r[i]$ is updated when $c[i] == Cmax$, where $Cmax$ denotes the maximum execution time of the tasks. When the job of task i is finished, it updates the relevant variables and sends an event $finish!$ to inform the scheduler of its termination($Run \rightarrow C5 \rightarrow Idle$). The automaton $v-task$ models the task under analysis. In contrast to $non-v-task$, $v-task$ resets the clock d and goes into location $Ready$ when a job is released. In the location $Ready$ and Run, when the condition $d == D[i]$ is satisfied, the task has missed its deadline and goes into the $Error$ location.

The automaton *scheduler* maintains the system state, allocates and schedules released jobs. When a task is released, an event $release!$ sent by T, and the transition $Init \rightarrow C1$ with $release?$ is taken. The value of $rlsT$ updated by T shows which task the released job belongs to. Then we check to see which processor is idle and record it ($p = PrmptW()$). If there is at least one idle processor ($p >= 1$), then the released job is allocated to it. After updating the relevant variables ($C1 \rightarrow C4$), an event $executeT!$ is issued ($C4 \rightarrow Init$) to inform the corresponding task to start execution. If there is no idle processor ($p == 0$), then the scheduler checks to see if there is any running job with lower priority than the newly-released job. If not, then the released job goes into the wait state ($C2 \rightarrow Init$); otherwise, the newly-released job ($C2 \rightarrow C3$) preempts the lower-priority job.

When a job finishes execution, an event $finish!$ is issued by $non-v-task$ or $v-task$, and the transition $Init \rightarrow C5$ with $finish?$ is taken. The value $fshP$ records which processor the finished job has been running on. If all jobs allocated to this processor have finished execution ($rn[p] == 0$), then the scheduler wakes up the waiting task with the highest priority if the wait queue is non-empty.

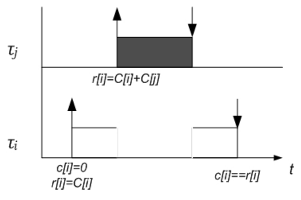


Fig. 2. Execution scenario for tasks i and j with $Prio(j) > Prio(i)$

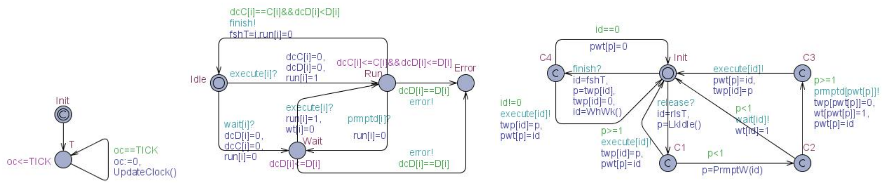
4 TA Model for Full Migration Scheduling

In contrast to restricted migration scheduling, a preempted job can resume execution on any available processor using full migration scheduling. Due to inherent limitations of the model-checking technology (we are not aware of any model-checkers that can handle fractional numbers.), we can only handle tasksets with integer task attributes. We can safely assume that a task's period and deadline to be integers, since it does not make a lot of sense to assign a non-integer period or deadline to a task from a real-time scheduling perspective, and it is almost never done in industry practice. However, it is possible for a task's execution time and deadline to be non-integers. We can round up the execution time to the nearest integer for schedulability analysis. We believe this is not a major limitation in practice. We can prove the following theorem (proofs omitted due to space limitations):

Theorem 1. *To determine schedulability of a periodic taskset whose attributes are all integers, it is sufficient to only consider task release times at integer time instants, which only produce execution traces in which all scheduling events (task release, preemption, blocking and finish) happen at integer time instants.*

Theorem 1 implies that the expressiveness of discrete time formalism is adequate for the purpose of schedulability analysis if we accept the limitation that all task attributes must be integers. Using the discrete time approach has the additional benefit of making it easier to model preemptive scheduling, since using a continuous time formalism would require a stopwatch mechanism to keep track of each task's execution time when it is preempted and resumed [13]. However, it is not necessarily true that using a discrete time approach always yields a smaller state space than using the continuous time approach, if there are long durations of time intervals within which no significant events happen.

Fig. 3(a) shows the automaton that generates periodic clock ticks. $TICK$ is a constant denoting granularity of clock ticks. When $oc == TICK$, the transition on edge $T \rightarrow T$ is taken, and all discrete clocks are incremented by 1 in the function $UpdateClock()$, which means that one digital clock tick has passed. Unlike continuous time clocks, the integer variable representing a discrete clock can be



(a) TA modeling the periodic clock tick

(b) TA modeling a task

(c) TA modeling the scheduler automaton

Fig. 3. TA model for full migration scheduling

paused or restarted. We use $dcC[i] = dcC[i] + run[i]$ to update $dcC[i]$ and in $UpdateClock()$. Setting $run[i] = 0$ pauses the discrete clock $dcC[i]$, and setting $run[i] = 1$ resumes it. Since the discrete clocks can be paused and resumed, we can model the time behavior of each task separately rather than accumulating the computing time of the jobs preempted it. As shown in Fig. 3(b), when the automaton is in location Run, if $dcC[i] == C[i]$, then task i has finished execution. When it is in location Run or $Wait$ and $dcD[i] == D[i]$, then task i has missed its deadline, and we determine the taskset to be unschedulable.

In contrast to restricted migration, the full migration scheduler maintains a global wait queue, in which all the ready jobs are waiting. As shown in Fig .3(c), when a job is released, the scheduler checks to see if there is an idle processor ($p = LkIdle()$). If yes ($p >= 1$), the job starts executing on it immediately ($C1 \rightarrow Init$). Otherwise, the scheduler checks to see if there is any running job can be preempted ($p = PrmptW(id)$). If yes ($p >= 1$), the newly-released job preempts the running job ($C2 \rightarrow C3 \rightarrow Init$). Otherwise, the released job waits ($C2 \rightarrow Init$). When some job is finished on a processor, the scheduler checks to see if there are any waiting jobs ($id = WhWk()$). If yes ($id! = 0$), the job with highest priority starts executing, otherwise the processor remains idle.

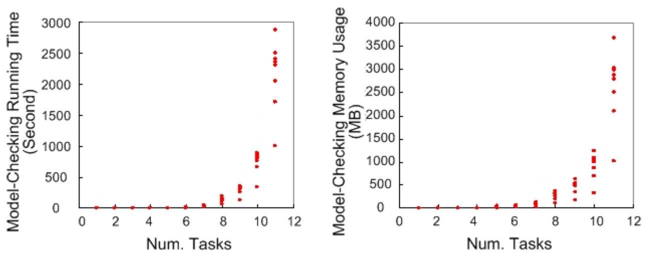
5 Performance Evaluation

We compare the schedulability analysis results of our method with classical methods. The model-checking experiments were run on a server with four AMD Opteron 844 (1.8GHz) CPUs and 8GB RAM running Fedora Linux. We use a utility program *memtime* developed by the UPPAAL group to record peak memory usage and running time of the model-checker. To our best knowledge, there is no known utilization bound test for the static-priority restricted-migration scheduling, so we only consider the case of full-migration scheduling in the following experiments.

We generated 200 tasksets, each consisting of 5 tasks running on 2 processors. Each task's period is chose randomly in the range of $[8, 20]$, and a task's execution time is the product of its period with a random value in the range of $[0.1, 0.5]$, rounded to the nearest integer. In the experience, 64 tasksets are accepted by Baker's test [16], 98 accepted by Andersson's test, 154 accepted by our method and 157 accepted by simulation in which all task release offsets are 0. We can see that the utilization bound tests in Baker and are indeed pessimistic and rejects a large number of tasksets that are actually schedulable. The acceptance ratio using simulation with zero task release offset is slightly larger than that using model-checking, as several tasksets are determined to be schedulable using simulation, but are in fact unschedulable since the worst-case response time for a task is maximized with some tasks have non-zero release offsets.

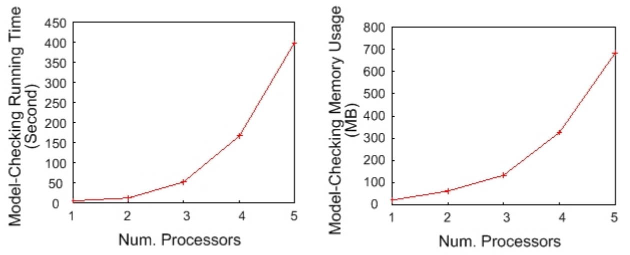
Next, we evaluate the performance and scalability[2]. Fig. 4 shows how the worst-case peak memory size and running time of UPPAAL increase with the

[2] Since the performance results are similar for restricted and full-migration scheduling, we only show the data for full-migration scheduling to give the reader a general idea of model-checking performance.



(a) Running Time (b) Peak Memory Usage

Fig. 4. UPPAAL performance for full-migration scheduling on 2 processors

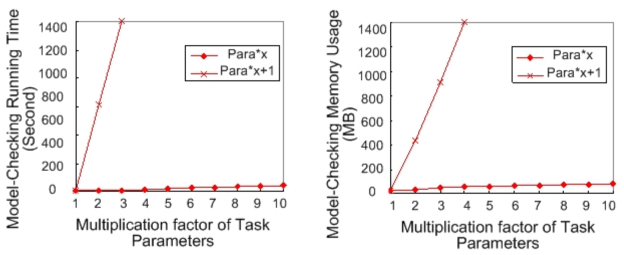


(a) Running Time (b) Peak Memory Usage

Fig. 5. UPPAAL performance for full-migration scheduling of 6 tasks

number of tasks with a fixed number of processors (2), and Fig. 5 shows how they increase with the number of processors with a fixed number of tasks (6).

We use a taskset with 6 tasks to show how model-checking complexity grows with scaling-up of taskset parameter values. The task parameters (period, deadline and execution time) in the first group are integer multiples of those of the original taskset, and the parameters in the second group are the integer multiples plus 1. A taskset in the second group are "pathological" in the sense that



(a) Running Time (b) Peak Memory Usage

Fig. 6. UPPAAL performance for full-migration scheduling with different scale factors of task parameters

task periods are relatively prime to each other, so the taskset has a very large hyper-period. The state space for a taskset in the second group grows up much faster with the scale factor than that for a taskset in the first group, which is confirmed by Fig. 6, where we can see that UPPAAL's performance deteriorates quickly with the increase in scale factor for the second group, while it stays more or less constant for the first group. On the other hand, adding 1 to each task's scaled execution time while scaling up its period and deadline has a negligible impact on performance. We can also see that UPPAAL handles long time durations gracefully as long as they are integer multiples of each other. In industry practice, task periods are typically assigned to be integer multiples of each other, thus making the model-checking approach more practically relevant.

6 Conclusions

In this paper, we use model-checking to provide an exact method to schedulability analysis of periodic tasksets on multi-processor systems, in order to overcome the pessimism of schedulability bound tests. As we can see in Section 5, the main limitation of the model-checking is state-space explosion, which limits the size of the problem that can be handled. This is especially problematic for real-time model-checkers like UPPAAL must handle continuous real-time clocks. As part of our future work, we plan to experiment with other modeling formalisms such as ACSR-VP [11], and compare their performance and scalability.

HW task scheduling on a FPGA shares many similarities with global task scheduling on identical multi-processors [3], where all processors in the system have identical processing speed and different task invocation instances may run on different processors. But it is actually a more general and challenging problem since a HW task may occupy a different area size on the FPGA while a SW task always occupies one and only one CPU. Some authors [14][15] have derived utilization bound tests for FPGA scheduling. We plan to apply model-checking to develop a schedulability analysis tool for FPGAs.

References

[1] Liu, C., Layland, J.W.: Scheduling Algorithms for Multi-Programming in a Hard Real-Time Environment. Journal of the ACM 20, 46–61 (1973)

[2] Lehoczky, J.P., Sha, L., Ding, Y.: The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior. In: IEEE Real-Time Systems Symposium (RTSS) (1989)

[3] Carpenter, J., Funk, S., Holman, P., Srinivasan, A., Anderson, J., Baruah, S.: A Categorization of Real-Time Multiprocessor Scheduling Problems and Algorithms, Handbook of Scheduling: Algorithms, Models and Performance Analysis, Chapman and Hall/CRC (2004)

[4] Amnell, T., Fersman, E., Mokrushin, L., Pettersson, P., Yi, W.: TIMES: a Tool for Schedulability Analysis and Code Generation of Real-Time Systems. In: FORMATS. International Workshop on Formal Modeling and Analysis of Timed Systems (2003)

[5] Goossens, J., Funk, S., Baruah, S.K.: Priority-Driven Scheduling of Periodic Task Systems on Multiprocessors. Real-Time Systems 25 (2003)
[6] Baker, T.P.: Multiprocessor EDF and Deadline Monotonic Schedulability Analysis. In: IEEE Real-Time Systems Symposium (RTSS), pp. 120–129 (2003)
[7] Baker, T.P.: An Analysis of EDF Schedulability on a Multiprocessor. IEEE Trans. Parallel Distrib. Syst. 16, 760–768 (2005)
[8] Bertogna, M., Cirinei, M., Lipari, G.: Improved Schedulability Analysis of EDF on Multiprocessor Platforms. In: Euromicro Conference on Real-Time Systems (ECRTS), pp. 209–218 (2005)
[9] Andersson, B., Baruah, S.K., Jonsson, J.: Static-Priority Scheduling on Multiprocessors. In: IEEE Real-Time Systems Symposium, pp. 193–202 (2001)
[10] Fersman, E., Mokrushin, L., Pettersson, P., Yi, W.: Schedulability analysis of fixed-priority systems using timed automata. Theor. Comput. Sci. 354(2), 301–317 (2006)
[11] Kwak, H.-H., Lee, I., Philippou, A., Choi, J.-Y., Sokolsky, O.: Symbolic Schedulability Analysis of Real-Time Systems. In: IEEE Real-Time Systems Symposium (RTSS) (1998)
[12] The UPPAAL Model-Checker, `http://www.uppaal.com`
[13] Cassez, F., Larsen, K.G.: The Impressive Power of Stopwatches. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 138–152. Springer, Heidelberg (2000)
[14] Danne, K., Platzner, M.: An EDF Schedulability Test for Periodic Tasks on Reconfigurable Hardware Devices. In: ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded (LCTES) (2006)
[15] Guan, N., Gu, Z., Deng, Q., Liu, W., Yu, G.: Improved Schedulability Analysis of EDF Scheduling on Runtime Partially Reconfigurable Hardware Devices. In: WDPRTS 2007 (2007)
[16] Baker, T.P.: An Analysis of Fixed-Priority Schedulability on a Multiprocessor. Real-Time Systems 32(1-2), 49–71 (2006)
[17] Baruah, S.K., Goossens, J.: Rate-Monotonic Scheduling on Uniform Multiprocessors. IEEE Trans. Computers 52(7), 966–970 (2003)