Schedulability Analysis of Global Fixed-Priority or EDF Multiprocessor Scheduling with Symbolic Model-Checking

Nan Guan¹, Zonghua Gu², Mingsong Lv¹, Qingxu Deng¹ and Ge Yu¹

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

Abstract

As Moore’s law comes to an end, multi-processor (MP) systems are becoming increasingly important in embedded systems design, hence real-time schedulability analysis for MP systems has become an important research topic. In this paper, we present an exact method for schedulability analysis of global multiprocessor scheduling with either Fixed-Priority (FP) or Earliest-Deadline-First (EDF) algorithms using the model-checker NuSMV. Compared to safe but pessimistic schedulability tests based on processor utilization bounds, model-checking can provide an exact answer to the schedulability of a taskset, as well as quantitative information on each task’s best-case and worst-case response times.

1 Introduction

As Moore’s law comes to an end, it becomes increasingly important to use multi-processor (MP) systems in embedded systems design. Therefore, real-time schedulability analysis for MP systems has become an important research topic. MP scheduling algorithms can be classified into three types based upon the permissible degree of inter-processor migration [1]:

- **No migration (partitioned).** The set of tasks is partitioned into as many disjoint subsets as there are processors available, and each such subset is associated with a unique processor. All jobs generated by the tasks in a subset must execute only upon the corresponding processor.
- **Restricted migration.** Each job must execute entirely upon a single processor. However, different jobs of the same task may execute upon different processors.
- **Full migration.** No restrictions are placed upon inter-processor migration.

Partitioned scheduling with a fixed 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, there are two methods to determine the schedulability of multiprocessor systems: theory proving, 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, i.e., all tasks are released at the same time. However, in contrast 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. We believe model-checking is a promising solution to this problem. Model-checking can be viewed as exhaustive simulation, exploring the entire system state space to prove or disprove properties, while simulation only explores part of the state space. Timed Automata based model-checking has been applied to single-processor scheduling and MP scheduling with restricted migration [2], but not to global MP scheduling due to difficulties in modeling task preemption and migration. In this paper, we use the symbolic model-checker NuSMV [3] to provide an exact method for schedulability analysis of global MP scheduling with either Fixed-Priority (FP) or Earliest-Deadline-First (EDF) algorithm. In addition to reducing pessimism, this technique has the additional benefit of being able to provide quantitative information of the minimum and maximum response times of each task.

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

2 Related Work

2.1 Utilization Bound Tests

For EDF-based MP scheduling, several authors have developed utilization bound tests. Goossens [4] developed a utilization bound test assuming that tasks have relative deadlines equal...
to the period. Baker [5] developed another utilization bound test that can handle relative deadlines less than or equal to the period. Baker [6] extended [5] 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 [7] presented an improved test, and showed that it is incomparable to [5], and in the sense that each test can accept tasks 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 [8] proved that the utilization guarantee for any static-priority MP scheduling algorithm, whether partitioned, restricted migration or full migration, cannot be higher than \((m+1)/2\) for an \(m\)-processor platform. For full-migration static priority scheduling, Andersson [8] defined a periodic taskset \(\{T_1, T_2, \ldots, T_N\}\), where each task’s deadline is equal to its period, to be a light system on \(m\) processors if it satisfies the following properties:

- \(\sum_{i=1}^{N} C_i \leq m^2/3m-2\)
- \(C_i \leq m^{2}/3m^{2}-2\), 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. In other words, if all individual utilizations are less than \(m^{2}/3m^{2}-2\) and the total utilization is less than \(m^{2}/3m^{2}-2\), then the taskset is schedulable on \(m\) processors using global RM.

Baker [9] presented a group of efficiently computable utilization bound tests for fixed-priority scheduling of periodic tasksets with arbitrary deadlines on a homogeneous MP system. 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 \(m(1-u_{max})/2 + u_{min}\).

2.2 Model-Checking for Schedulability Analysis

TIMES [10] is a tool for schedulability analysis for single processor systems based on UPPAAL. Krcal [2] showed that the schedulability analysis problem for restricted-migration MP scheduling is decidable for systems with non-preemptive schedulers or tasks with fixed execution times, but undecidable for systems with variable task execution times and a preemptive scheduler. In this paper, we address preemptive MP scheduling with full migration. Unlike the continuous time modeling formalism of Timed Automata, the discrete time modeling formalism of NuSMV does not have any decidability issues.

In [11], we used UPPAAL to solve the MP schedulability analysis problem. In order to model preemptive scheduling, we imposed a discrete global clock tick that synchronizes all task execution and release events in the system. This approach converts the continuous-time formalism of Timed Automata into a discrete-time formalism, which really runs counter to the spirit of Timed Automata and causes an exponential state space increase with the size of timing parameters. In this paper, we use the untimed model-checker NuSMV by treating each state transition in NuSMV as a time step. Our experience shows that NuSMV generally exhibits better scalability than UPPAAL, with the additional benefit of being able to calculate quantitative information of the minimum and maximum response times of each task, not just a binary yes/no answer of schedulability. In [12], we used HyTech [13], a model-checker for Linear Hybrid Automata, for single-processor schedulability analysis with quantitative information, with the additional benefit of using HyTech’s parametric analysis capability to derive task parameters such as the critical scaling factor without using binary search. However, HyTech is not being actively maintained, and it is not as scalable as UP- PAAL or NuSMV.

Verus [14] is a discrete-time model-checker for simple timed systems. The quantitative analysis algorithms in Verus has been incorporated into the NuSMV model-checker in the form of \(\min\) and \(\max\) operators, which are used in this paper.

There are two possible approaches to modeling real-time scheduling with NuSMV. The traditional approach is to use a global synchronizing clock tick. Another approach is to adopt the variable time advance from discrete event simulation, where an integer is used to keep track of the global time, which is incremented by a variable step size at each state transition based on the nearest future event (task release, task finish, etc.). We have implemented both approaches. Surprisingly, even though the VTA approach needs fewer state transition steps to reach the same final state than the global clock tick approach, it actually exhibits worse scalability, which is likely due to increased model complexity. Therefore, we focus on the global clock tick modeling approach in this paper.

3 Modeling Global MP Scheduling with NuSMV

We consider a homogenous multiprocessor system with \(M\) processors. A taskset consists of \(N\) periodic tasks. Each task \(T_k = (C_k, D_k)\) is characterized by its execution time \(C_k\) and period \(D_k\), which is equal to its deadline. Task \(T_k\) consists of a sequence of jobs. At a given time instant, \(C_k\) represents the cumulative processor time that has been granted to the current job, and \(D_k\) represents the wall-clock time that has elapsed since the release time of the current job. \(T_k\)’s remaining execution time is \(C_k - c_k\), and its time-to-deadline is \(D_k - d_k\). The task \(T_k\) is alive if the current job hasn’t finished execution \((c_k < C_k)\), and is sleeping if the current job has finished \((c_k = C_k)\).

An important consideration is encoding of the processor granting to tasks. At each time instant, the OS scheduler needs to decide which task should run, and on which processor the chosen task should run. It turns out to be unnecessary to explicitly model which processor a task runs on in the NuSMV model due to the following property of global scheduling on homogeneous multi-processors (proof omitted due to space limitations):

Theorem 1 Consider a taskset scheduled by a global MP scheduling algorithm. At any time, if we swap two running tasks’ processor assignments (assuming zero task migration overhead), the taskset’s schedulability is not affected.

Therefore, we only need to model which tasks are currently running, and make sure that the number of currently running tasks is not larger than the number of processors. This observation greatly simplifies the model.
3.1 Modeling Global FP Scheduling

Algorithm 1 shows the NuSMV model of global FP scheduling for the taskset in Table 1 on two processors:

Table 1. Another example taskset with rate monotonic priority assignment. Smaller priority value denotes higher priority.

<table>
<thead>
<tr>
<th>Task</th>
<th>Period</th>
<th>Comp. Time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>6</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>t2</td>
<td>8</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>t3</td>
<td>10</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>t4</td>
<td>12</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

Algorithm 1 NuSMV model for global FP scheduling.

```plaintext
MODULE main
VAR
c1:0..2; c2:0..3; c3:0..2; c4:0..4;
d1:0..6; d2:0..8; d3:0..10; d4:0..12;
DEFINE
C1:=2; C2:=3; C3:=2; C4:=4;
D1:=6; D2:=8; D3:=10; D4:=12;
PN:=2;
T1_rq := ci < Ci;
T1_lv := ci < Ci & di < Di - 1;
T1_lv_rls := ci = Ci & di < Di - 1;
T1_slp_rls := ci = Ci & di = Di - 1;
T1_miss := Di - di < Ci - ci;
T1_slp := (ci < Ci & di < Di - 1);
T1_rls := (ci < Ci & di = Di - 1);
T1_lv_rls := ci < Ci & di < Di - 1;
T1_lp := ci < Ci & di < Di - 1;
T1_gp := T1_rq;
T2_gp := T2_rq & (T1_rq < PN);
T3_gp := T3_rq & (T2_rq < PN);
T4_gp := T4_rq & (T3_rq < PN);
ASSIGN
init(c1):=2; init(c2):=3;
init(c3):=2; init(c4):=4;
init(d1):=0..5; init(d2):=0..7;
init(d3):=0..9; init(d4):=0..11;
next(c1) := case
  T1_lv & T1_hp : ci + 1;
  T1_lv_rls & T1_hp | T1_slp_rls : 0;
  T1_slp | T1_lp & T1_hp : ci + 1;
  1 : 0;
esac;
next(d1) := (d1 + 1) mod D1;
-- Similarly for tasks 2, 3 and 4.
DEFINE
MISS := T1_miss|T2_miss|T3_miss|T4_miss;
SPEC AG !MISS
COMPUTE MIN[di=0 & ci=0, ci=C1 - 1 & T1_hp]
COMPUTE MAX[di=0 & ci=0, ci=C1 - 1 & T1_hp]
```

The VAR section defines the variables representing the remaining execution time (ci) and the time-to-deadline (di) of each task t_i. The range of ci is from 0 to t_i's execution time C_i, and the range of di is from 0 to t_i's period/deadline D_i.

In the beginning of the DEFINE section, the execution time and period (relative deadline) of each task and the number of processors (PN) are defined, followed by the definitions of the task states:

- T1_rq := ci < Ci: τ_i is alive.
- T1_lv := (ci < Ci & di < Di - 1): τ_i is alive and its remaining execution time and time-to-deadline are both greater than 1, so a new job will not be released at the next time instant.
- T1_slp := (ci = Ci & di < Di - 1): τ_i has finished execution, and a new job will not be released at the next time instant.
- T1_rls := (ci = Ci & di = Di - 1): Task τ_i has finished execution, and a new job will be released at the next time instant.
- T1_miss := Di - di < Ci - ci: τ_i's remaining execution time is larger than its time-to-deadline, so it will miss deadline.

We distinguish between the states with di < Di - 1 (T1_lv, T1_slp) and those with di = Di - 1 (T1_rls, T1_gp), since if a task's time-to-deadline is 1, the new job of the task will be released at the next time instant, when ci becomes Ci. This discontinuous jump of ci's value needs to be explicitly encoded in the model.

T1_gp = 1 means that Ti is a granted processor, while T1_gp = 0 means that Ti is not granted a processor. For task T0 with the highest priority, it will be granted a processor as soon as it makes a request. For any other task Ti, it will be granted a processor if it makes a request and the number of tasks with higher priority and making requests for processors is smaller than PN, as we mentioned earlier.

In the ASSIGN section, all the variables ci and di are initialized. To model all possible release offsets in the range [0, Di), for each task Ti, di is initialized to be a non-deterministic number in the range [0, Di), which forces the model-checker to exhaustively try all possible release offsets in the range [0, Di) during state space exploration. This is necessary for MP scheduling in order to find the worst-case task phasing resulting in the largest task response times, as we explained in Section 1.

In the following, the state transition are defined:

- A task is initially in the T1_lv state when it is first released. At any given time instant,
  - If it is granted a processor, it either takes the run2 transition to enter T1_slp at the next time instant if it has only 1 unit of remaining execution time; or it takes the run3 transition to enter T1_lv if its time-to-deadline is 2; or it takes the run0 transition to stay in state T1_lv.
  - If it is not granted a processor, it either takes the wait2 transition to enter the T1_miss state if its time-to-deadline will be smaller than its remaining execution time at the next time instant; or it takes the wait1 transition to stay in state T1_lv.

- In the T1_lv_rls state, at any given time instant.
  - If the task is granted a processor, it takes the run+release transition, which means both its remaining execution time and time-to-deadline will become 0, and a new job will be released in the next time instant.
  - If it is not granted a processor, it takes the wait3 transition to enter T1_miss.

- In the T1_slp state, it may take the sleep1 transition to stay in T1_slp if di < Di - 2, or the sleep2 transition to enter T1_slp_rls if di = Di - 2.

- There is only one transition sleep+release from state T1_slp_rls when a new job is released, and it enters the T1_lv state.

558
Finally, we define the condition MISS denoting a deadline miss of any task, and ask NuSMV to check the CTL formula AG !MISS, which means that a deadline miss never occurs. If the formula is proven true, then the taskset is schedulable; otherwise it is non-schedulable, and NuSMV provides a counterexample as an execution trace leading to the the deadline miss. In addition to obtaining a yes/no answer, we can also use the built-in commands \( \text{MIN}[a,b] \) and \( \text{MAX}[a,b] \) to compute the best and worst case response times of each task. \( \text{MIN}[a,b](\text{MAX}[a,b]) \) counts the minimum(maximum) number of state transitions from state \( a \) to state \( b \). In the state satisfying the predicate \((\text{di}=0 \& \text{ci}=0)\), task \( i \) is released. In the state satisfying the predicate \((\text{ci}=\text{Ci}-1 \& \text{Ti}_{hp})\), task \( i \)'s remaining execution time is 1 and it is granted a processor to execute, so it will finish execution at the next time instant, and \( \text{ci} \) will be reset to \( \text{Ci} \). So

\[
\text{MIN}[\text{di}=0\&\text{ci}=0,\text{ci}=\text{Ci}-1\&\text{Ti}_{hp}] + 1
\]

is the BCRT and

\[
\text{MAX}[\text{di}=0\&\text{ci}=0,\text{ci}=\text{Ci}-1\&\text{Ti}_{hp}] + 1
\]

is the WCRT of task \( \tau_i \).

### 3.2 Modeling Global EDF Scheduling

The NuSMV model for global EDF is similar to that for FP, except that each task’s priority is assigned dynamically based on its time-to-deadline at runtime. Algorithm 2 shows the NuSMV model for the taskset in Table 1 on 2 processors. At each time instant, the scheduler should select \( PN \) tasks (if there are more than \( PN \) tasks that are alive) with highest priority to execute.

We use \( \text{sij} := si > sj \) to denote the result of priority comparison between tasks \( \tau_i \) and \( \tau_j \), where \( si = Di - di \). If \( \text{sij} \) and \( \text{Tj}_{rq} \) are both true, then \( \tau_j \) has not finished execution of its current job, and it has higher priority than \( \tau_i \). So for \( \tau_i \), if the following holds:

\[
(\text{Ti}_{rq} \& \sum_{j\neq i} \text{sij} \& \text{Tj}_{rq} < \text{PN}) = 1
\]

then \( \tau_i \) can be granted a processor for execution. Since \( \text{sij} = \neg \text{sji} \), we only need to define \( \text{sij} \) for the case of \( j>i \), as we can use \( \neg \text{sji} \) to represent \( \text{sij} \) when \( i<j \).

**Algorithm 2** NuSMV Model for global MP EDF Scheduling.
(Only the part that is different from FP scheduling is shown.)

```
DEFINE
  s1s2:=s1>s2; s1s3:=s1>s3; s1s4:=s1>s4;
  s2s3:=s2>s3; s2s4:=s2>s4; s3s4:=s3>s4;

  TL_qp := Tl_rq &
  ((s1s2&T2_rq)+(s1s3&T3_rq)+(s1s4&T4_rq)<PN);
  T2_qp := T2_rq &
  ((t1s2&T1_rq)+(s2s3&T3_rq)+(s2s4&T4_rq)<PN);
  T3_qp := T3_rq &
  ((t1s3&T1_rq)+(s1s3&T3_rq)+(s3s4&T4_rq)<PN);
  T4_qp := T4_rq &
  ((t1s4&T1_rq)+(s1s4&T4_rq)+(s3s4&T3_rq)<PN);
```

### 4 Performance Evaluation

We measure performance of model-checking with two metrics: accuracy of its schedulability analysis decisions compared to utilization bound tests, and scalability of the model-checker itself in terms of peak memory size and running time of each model-checking session. The accuracy metric is measured with the acceptance ratio, i.e., the percentage of tasksets determined to be schedulable by either model-checking or utilization bound tests. For the scalability metric, we use a utility program memtime [15] to record peak memory usage and running time of each model-checking session. The experiments were performed on a Linux workstation with Intel Core2 Duo 2.4GHz CPU and 4GB RAM.

![Figure 1. Acceptance ratios.](image)

We generated 1000 random tasksets, each consisting of 8 tasks running on 4 processors. Each task’s period is chosen randomly in the range of [8, 20]. As with any modeling approach based on global clock ticks, the state space increases exponentially with the number of time steps, so this approach is sensitive to the absolute value of timing parameters. We set the maximum size of the timing parameters to be 20 in our experiments. One way to handle larger timing parameters is to scale all numbers by a constant factor with a loss of precision, e.g., divide all numbers by 10 and discard the fractional parts. Figure 1-(a) compares the acceptance ratios of the utilization bound tests and model-checking for Rate Monotonic scheduling, a widely used FP scheduling algorithm where tasks with smaller periods are assigned higher priorities. The line labeled BAK represents acceptance ratios of the utilization bound test proposed by Baker [9]; the line labeled ABJ represents results from the utilization bound test proposed in [8]; the line labeled MC represents results from model-checking. Figure 1-(b) shows similar information for EDF scheduling. The line labeled GFB represents results from the utilization bound test in [4], the line labeled BCL represents results from the utilization bound test in [7]. We can see that the utilization bound tests are indeed safe but quite pessimistic compared with the exact analysis.
technique using model-checking, and declares many schedulable
tasksets to be non-schedulable.

![figure 2](image)

Figure 2. Peak memory size and running time of each model-checking session for global FP scheduling.

In order to measure scalability of model-checking, Figure 2 shows how the peak memory size and running time of model-checking vary with the number of tasks \( M \) for global FP scheduling, where \( M = N - 1 \), \( N \) is the number of processors. Model-checking for EDF scheduling (performance data not shown here due to space limitations) generally incurs larger peak memory sizes than those for FP scheduling, which may be due to the added complexity at each time step to select the task with nearest deadline. Since NuSMV uses symbolic representation of the state space using the OBDD (Ordered Binary Decision Diagram) data structure, memory size of the model-checker does not necessarily increase with the increase of the state space size, but may be dependent on a number of additional factors including the shape and regularity of the state space. Increased complexity in the model may lead to increased irregularity of the state space, and in turn increased peak memory size of model-checking.

5 Conclusions

In this paper, we use the symbolic model-checker NuSMV to provide an exact method to determine schedulability of a periodic taskset on a multi-processor system in order to avoid the pessimism of schedulability bound tests, as well as to obtain quantitative response time information.

References


