Exact Schedulability Analysis of Global Multi-Processor IUF Scheduling using Symbolic Model Checking

Alok Lele, Ajith Kumar, Rohith H.M. and Sanjay Singh
(alok.lele, aji3003, rohithhm, sanjaysingh.ict)@gmail.com
Department of Information and Communication Technology,
Manipal Institute of Technology, Manipal, India

***Abstract*—Real time scheduling spans a broad spectrum of algorithms for simple uniprocessors to highly sophisticated multi-processors. As the world nears the end of Moore's Law, more emphasis is being laid on the use of multi-processors in embedded systems. This has lead to a rise in several new scheduling schemes for multi-processor environments. Naturally there arises a need for an accurate comparison of the schedulabilities of each of these schemes in order to prove their superiority over others. The two-fold aim of this paper is to present a technique for the exact schedulability analysis of global multi-processor schedulers and, based on this intermediary result, prove the schedulability superiority of a novel multi-processor scheduling algorithm. The exact schedulability analysis technique presented here is symbolic model checking in NuSMV, and we have used this technique to investigate the schedulability of global Instantaneous Utilization Factor (IUF) based scheduling. Our analysis shows a substantial improvement in the acceptance of task sets with heavier utilizations.**

***Index Terms*—Symbolic Model checking, Schedulability, IUF (Instantaneous Utilization Factor) scheduling, Multi-processor scheduling.**

I. Introduction

As we near the end of Moore's Law [1], the next feasible option left to satisfy the growing need of processing capacity is to use multi-processors in embedded system designs. The problem with multi-processor (MP) scheduling is to devise an algorithm to exploit the maximum capacity of each processor for efficient processing. As a result there has been an uprise in new scheduling schemes for multi-processor environments. Schedulability of these proposed schemes has been the primary factor in deciding the superiority of one scheduling algorithm over others. Presently there are several methods for schedulability analysis.

There has been a number of utilization bound tests (UBTs) proposed for the schedulability analysis of various schedulers. Some of these have been stated in *Goossens et al.* [2], *Bertogna et al.* [3], *Andersson et al.* [4]. The problem with all these tests is that they are safe but pessimistic in nature i.e. they output false for many tasksets that in actual can be scheduled using the scheme it has been tested for. Another approach would be to simulate the schedule. But simulation proves to be ineffective for two main reasons. Firstly, it is unsafe since it explores only one execution trace instead of an exhaustive exploration. Secondly, a widely adopted convention for simulation is to set the phase of all tasks to zero i.e., all tasks are released at the same time. Contradictory to single processors this may not prove to be the worst case in multi-processor environments.

Keeping this in mind, the primary aim of this paper is to propose an exact schedulability analysis technique for global multi-processor scheduling schemes. Such a technique should be able to give an accurate and non-pessimistic result about the feasibility of a scheduling algorithm for any given taskset. Such a technique is devised using symbolic model checking using NuSMV. The second objective of this paper is to use this method to compare the schedulability of two widely known MP scheduling schemes, namely *Rate Monotonic* (RM) and *Earliest Deadline First* (EDF) with a novel scheduling algorithm based on the *Instantaneous Utilization Factor* (IUF) [5] of the tasks in the system. The algorithm under observation was proposed by *Naik et al.* [5] for single processor scheduling. This paper introduces the advantages of this algorithm in a multi-processor environment also.

We have adopted model checking as a promising accurate technique for the purpose of schedulability analysis. Model-checking can be viewed as exhaustive simulation, exploring the entire system state space to prove or disprove properties. We use the discrete time model checker, NuSMV [6] in this paper. In addition to reducing pessimism, NuSMV can also provide quantitative information of the minimum and maximum response times of each task apart from a simple yes/no answer about the feasibility of a taskset.

The paper is hereforth organized as follows. Section II discusses about the motivation and related works for this paper. Section III describes the proposed scheduling scheme along with the technique adopted for the exact schedulability analysis. Section IV describes the performance evaluation of the proposed scheme as against other well known MP scheduling schemes. Finally the future scope and conclusion has been drawn in Section V.

II. Motivation and Related Works

A. Multi-Processor Scheduling

The problem of real time scheduling spans a broad spectrum of issues pertaining to single as well as multi-processor environments. Accordingly a scheduling requirements of an algorithm can be stated as:

- It should guarantee that all hard real time tasks must meet their deadlines
- It should have a high degree of schedulable utilization
- It should be able to accommodate aperiodic and sporadic tasks
- It should have low scheduling overheads (E.g. context-switching)
- It should have low average response times.

978-1-4244-6653-5/10/$26.00 ©2010 IEEE

We will be addressing these various performance parameters along with some others for evaluating our algorithm as against other MP scheduling schemes.

Classification of multiprocessor systems, according to *Carpenter et al.* [7], can be done as follows:

- *No Migration (partitioned):* The set of tasks is partitioned into a number of disjoint subsets equal to the number of processors available, and each 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:* There is no restrictions upon interprocessor migration. Task can be pre-empted and sent, at some later time, to another processor to be completed.

Henceforth we will be concerned with only with uniform multi-processor scheduling environments with full migration since it is most applicable to the scheduling scheme at hand.

B. Schedulability analysis

1) Utilization Bound Tests: Utilization bound tests were first proposed by Goossens et al. [2] in which it is assumed that tasks have relative deadline equal to their period. Baker [3] slightly modified the assumption such that utilization bound test can be performed on tasks with relative deadline less than or equal to their period. Baker derived simple sufficient conditions for schedulability of systems of periodic or sporadic tasks in a multiprocessor preemptive scheduling environment. Andersson et al. [4] proved that for fixed-priority scheduling, the utilization guarantee whether partitioned, restricted migration or full migration, cannot be higher than $(m+1)/2$ for an $m-$processor platform. Andersson defined that a periodic task system τ is light on m processors if it satisfies the following properties:

$$u_i \leq \frac{m}{3m-2} \ , \forall i \ \tau_i \in \tau \tag{1}$$

$$U(\tau) \leq \frac{m^2}{3m-2} \tag{2}$$

where u_i is the utilization of the task τ_i, $U(\tau)$ is the total utilization of the taskset and m is the number of processors in the system.

Andersson showed that any periodic task system τ that satisfies the above two conditions on m processors will be scheduled to meet all its deadlines by RM algorithm [4].

Baker [3] showed a group of efficiently computable utilization bound tests for fixed-priority scheduling of periodic tasksets with arbitrary deadlines on a homogeneous MP system. In case of a special scenario when deadline equals period and priorities are RM, any set of tasks with maximum individual task utilization u_{max} and minimum individual task utilization u_{min} is feasible as long as:

$$U(\tau) \leq \frac{m(1-u_{max})}{2} + u_{min} \tag{3}$$

Efficiently computable schedulability tests for EDF and DM (Deadline Monotonic) scheduling on a homogeneous multiprocessor system which allow pre-period deadlines were demonstrated in [3].

2) Schedulability Analysis Using Model Checking: There are a number of tools available for schedulability analysis using model checking. One of the widely used tool is UPPAAL. UPPAAL is an integrated tool environment for modeling, validation and verification of real-time systems modeled as networks of timed automata, extended with data types (bounded integers, arrays). The problem with UPPAAL was that it was based on discrete global clock tick that synchronizes all task execution and release events in the system. So if the hyper period of a taskset is huge then it will be laborious to model it. Kracal [8] showed that scheduling is decidable for systems with restricted-migration MP scheduling with non-preemptive schedulers or tasks with fixed execution times, but undecidable for systems with variable task execution times and a preemptive scheduler. Guan et al. [9] used UPPAAL to model preemptive scheduling by imposing 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 UPPAAL and causes an exponential state space increase with the size of timing parameters. Guan et al. [1] uses the untimed model checker NuSMV by treating each state transition in NuSMV as a time step. NuSMV generally exhibits better scalability than UPPAAL, and it has 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 [1]. Hence NuSMV would be more preferable than UPPAAL.

NuSMV can be modeled either by using a global synchronizing clock tick or by adopting the variable time advance from discrete event simulation, where an integer is used to keep track of the global time. This integer which is incremented by a variable step size at each state transition based on the nearest future event. Both the approaches have been successfully implemented in this paper for scheduling EDF, RM and IUF scheduling algorithms.

C. IUF Scheduling

Naik et al. [5] proposed a pre-emptive dynamic priority scheduling algorithm based on the instantaneous utilization of the task. Instantaneous Utilization Factor (IUF) is the processor utilization of the task at any instant and calculated as the ratio between the remaining execution time and the time to the end of its current period. IUF has several interesting characteristics such as its high utilization bound, dynamic predictability and easy accommodation of sporadic and aperiodic jobs. In the case where the deadline of the task is equal to its period, the instantaneous utilization factor of a task can be represented as:

$$\mu_i = \frac{\text{Remaining execution time}}{\text{Time to deadline}} \tag{4}$$

IUF schedules a particular job for a fixed duration of time. At the end of each time duration, the instantaneous utilization of the tasks are computed and the most urgent tasks are sent to the scheduler. In this paper we have extended the concept of IUF scheduling from a single processor to a multi-processor platform.

III. Global IUF scheduling and NuSMV based Schedulability Analysis

A. Global IUF Scheduling

As mentioned in the earlier section, Global IUF Scheduling is an improvement on [5]. To shift IUF scheduling onto a multi-processor platform requires various considerations to be made. We shall discuss each of the issues one at a time.

1) Runtime Queuing: This technique was proposed in [5] to overcome the problem of excessive context switching in the basic IUF scheduling by recording the activity of the first hyper period and re-ordering its sequence for all the future hyper periods. To facilitate task shuffling on multiprocessors we can make use of multiple run-time queues, one for each processor. But this will increase the overheads of maintaining additional runtime queues which are quite expensive especially in the domain of embedded applications where processing capabilities are themselves limited. The other solution could be to use a single runtime queue and log the context-switchings along with their processor ID's. This will increase the length of a single queue but effectively decrease the total overhead by approximately $\frac{1}{m}$, where m is the number of processors.

2) Task Shuffling: Task shuffling involves the actual re-distribution technique facilitated by run-time queuing. The concept can be summarized as grouping of task-slices of the same task between two successive task arrivals. Implementation of task shuffling on multi-processors can be done with ease for *restricted migration* or *no migration* just like it was done on single processor systems. Task shuffling for such environments can be done independently for each processor. In the case of *full migration*, the global knowledge of the task-slices that are executing on each of the processors is required. This can be done by:

- First, group successive execution of the same job of different processors
- Next, perform regular shuffling procedure to each processor keeping in mind that jobs are not allowed to run in parallel.

B. Modeling Global IUF

Following assumptions has been considered before we start modeling of global IUF for a multi-processor platform:

- System is a uniform multi-processor system with each processor having 100% processing capacity
- The deadlines of the tasks are equal to its periods
- The system allows full migration of tasks.

Modeling the global IUF scheduler involves developing a Kripke structure of how the taskset is executed in accordance to the behavior defined by IUF scheduling. It is required to define the following :

- The instantaneous values of each task parameter
- The state of the system at that time instant
- The next transition decision.

By instantaneous values of the system we describe the amount of execution time completed c_i and the amount of time elapsed after the release of the task d_i. These parameters will help us evaluate properties of the system that define its current state. Modeling a state of the system is described in **Algorithm 1**. These properties are calculated for each state generated in the model.

Algorithm 1 Modeling a state of the system

```
for i ← 1 to n do
    c_i := 0;
    d_i := 0;
end for
for each Task do
    t_i_lv := TruthValue(c_i < C_i AND d_i < D_i)
    t_i_rls := TruthValue(c_i = C_i AND d_i = D_i − 1)
    t_i_miss := TruthValue(D_i − d_i < C_i − c_i)
    t_i_util := (C_i − c_i)/(D_i − d_i)
end for
```

Each state has been described using a set of properties as follows:

- The function *TruthValue()* is a boolean function that returns whether the predicate is mentioned as parameter is true or not
- t_i_lv : describes whether execution of a task is remaining (alive) or not. A task is alive if its completed execution time is less than its total execution time and if its not yet reached its deadline
- t_i_rls : describes the condition wherein the next job of task is going to be released at the next time instant
- t_i_miss : describes whether a task will miss its deadline or not. A task will miss its deadline if at any instant its remaining execution time exceeds its time to deadline. The use of this state is important to model check if the task misses its deadline during system execution
- t_i_util : computes the instantaneous utilization of the task computed as the ratio of the remaining execution time to the remaining time to deadline.

Modeling the transition of the system i.e. the actual execution of the system for each duration of time has been described in **Algorithm 2**. This algorithm can be generically used for modeling the transition of any scheduling algorithm since the algorithm only describes which task is granted the processor and how the instantaneous task parameter value changes. More particularly a task will execute only if it has not completed execution and is granted a processor. If the next job of the task is set to be released at the next time instant then the value of the task's execution time c_i should be reset. The function *GrantProcessor(t_i)* returns *true* if the task t_i will be granted a processor to execute on in the next time instant. This can be represented logically by the predicate:

$$t_i\_lv \;\&\; \left(\sum_{j \neq i} (\mu_j > \mu_i \& t_j\_lv) < m \right) \tag{5}$$

where m is the number of processors. Equation 5 describes the predicate such that if the number of tasks whose instantaneous utilization is less than the number of available processors, then this predicate returns *true* and the task t_i is granted the processors.

Once the system has been modeled it can be checked for the CTL specification such as:

AG !MISS

This enables us to check the model for the occurrence of any missed deadline in our task system. If the system does not miss any deadline then the output of this CTL specification is *true*.

Algorithm 2 Defining transition between states

```
for Each Task do
    d_i := d_i + 1
    switch
    case t_i_lv and GrantProcessor(t_i):
            c_i := (c_i + 1)mod(C_i)
    case t_i_slp_rls:
            c_i := 0;
    end switch
end for
```

Otherwise, the output is *false* and along with it the NuSMV Model Checker will produce a counter example demonstrating how the deadline was missed.

NuSMV provides the additional functionality of calculating MIN and MAX lengths between any two states in the entire model. Based on this we compute the minimum, maximum and average response times of the tasks. Another additional benefit of NuSMV is that it can accommodate all possible phases of each task set in a single model providing an accurate worst-case analysis for any given taskset.

IV. Performance Evaluation

The performance of schedulability analysis through model checking has been clearly verified in [1] which clearly indicates the accuracy of model checking as compared to utilization bound tests such as those mentioned in [2], [3], [4]. On this foundation, the performance evaluation of global IUF scheduling has been done in comparison with RM and EDF on two parameters, namely *acceptance ratio* and *average response time*.

Fig. 1 indicates the acceptance ratios of global IUF in comparison with RM and EDF. It is clearly visible that the schedulability bound of IUF scheduling is substantially greater than those of EDF and RM. This indicates that IUF scheduling has the capacity to schedule heavier tasksets.

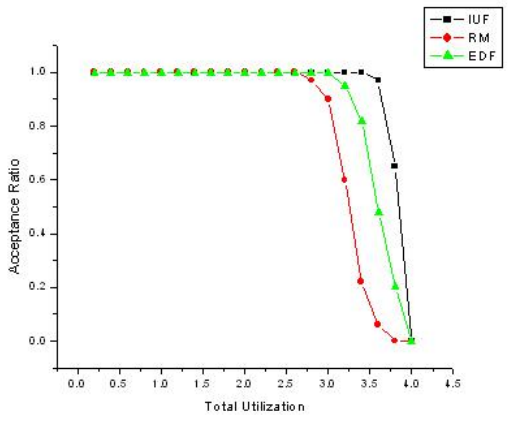

Fig. 1. Acceptance Ratio Graph

The average response time along with some other parameters have been displayed in Table I. The average response time of IUF is observed to be remarkably lower than the average response time of EDF and RM. This has also been computed within the NuSMV model checker by exploiting its additional quantificative functionalities.

If *slack stealing* mechanism [10] is incorporated into the IUF framework we can improve upon sustainability in a sporadic environment as well. The high context switching is the observed cost of improved schedulability which more than compensates for it.

TABLE I
Performance Analysis

Parameters	EDF	RM	IUF
Average Response Time	MEDIUM	HIGH	LOW
Sustainability	LOW	LOW	MEDIUM
Context Switching	MEDIUM	LOW	HIGH

V. Conclusion and Future Work

The observed shift from use of single processors to multi-processors in real-time systems and the consequent uprise in the development of multi-processor scheduling schemes have lead to the need of an accurate schedulability analysis technique.

In this paper we have discussed the extension of the single processor IUF model into a multi-processor environment. We have also shown how the model checking can be used for the exact schedulability analysis of such scheduling algorithms. We have exploited the accuracy and exhaustive state space exploration mechanism of symbolic model checking to validate the high schedulability bound of global IUF scheduling as against well known MP scheduling schemes. NuSMV also enabled us to determine the average response times of the tasksets for each of the scheduling schemes discussed. Therefore we can conclude that global IUF scheduling is a stable solution for multi-processor scheduling environments.

As part of future work, we plan to address optimization issues pertaining to global IUF scheduling as well as address issues of more general task models. Some method has to be devised for those tasksets which have their total utilization less than the number of processors available. A count for the inter-process communication can also be taken into consideration.

References

[1] N. Guan, Z. Gu, M. Lv, Q. Deng, and G. Yu, "Schedulability Analysis of Global Fixed-priority or EDF Multiprocessor Scheduling with Symbolic Model-Checking," *Object-Oriented Real-Time Distributed Computing, IEEE International Symposium on*, pp. 556–560, 2008.

[2] J. Goossens, S. Funk, and S. Baruah, "Priority-Driven Scheduling of Periodic Task Systems on Multiprocessors," *Real-Time Syst.*, vol. 25, no. 2-3, pp. 187–205, 2003.

[3] T. P. Baker, "Multiprocessor EDF and Deadline Monotonic Schedulability Analysis," *Real-Time Systems Symposium, IEEE International*, p. 120, 2003.

[4] B. Andersson, S. Baruah, and J. Jonsson, "Static-priority scheduling on multiprocessors," University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, Tech. Rep., 2001.

[5] R. Naik, V. Joshi, and R. R. Manthalkar, "IUF Scheduling Algorithm for Improving the Schedulability, Predictability and Sustainability of the Real Time System," *International Conference on Emerging Trends in Engineering and Technology*, pp. 998–1003, 2009.

[6] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "NuSMV 2: An OpenSource Tool for Symbolic Model Checking," in *CAV '02: Proceedings of the 14th International Conference on Computer Aided Verification*. London, UK: Springer-Verlag, 2002, pp. 359–364.

[7] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah, "A Categorization of Real-time Multiprocessor Scheduling Problems and Algorithms," in *Handbook on Scheduling Algorithms, Methods, and Models*. Chapman Hall/CRC, Boca, 2004, p. 30:130:19.

[8] P. Krc'al, M. Stigge, and W. Yi, "Multi-processor Schedulability Analysis of Preemptive Real-Time Tasks with Variable Execution Times," in *FORMATS*, 2007, pp. 274–289.

[9] M. Bertogna, M. Cirinei, and G. Lipari, "Improved Schedulability Analysis of EDF on Multiprocessor Platforms," in *ECRTS '05: Proceedings of the 17th Euromicro Conference on Real-Time Systems*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 209–218.

[10] J. P. Lehoczky and S. R. Thuel, "Scheduling periodic and aperiodic tasks using the slack stealing algorithm," *ACM Transactions*, pp. 175–198, 1995.