International Journal of Reconfigurable and Embedded Systems (IJRES)
Vol. 8, No. 2, July 2019, pp. 114~123
ISSN: 2089-4864, DOI: 10.11591/ijres.v8.i2.pp114-123

Performance optimization of task intensive real time applications on multicore ECUs-a hybrid scheduler

Geetishree Mishra[1], Rajeshwari Hegde[2]
[1]Department of Electronics & Communication Engineering, BMS College of Engineering, India
[2]Department of Telecommunication Engineering, BMS College of Engineering, India

Article Info

Article history:

Received Feb 9, 2019
Revised Apr 21, 2019
Accepted May 1, 2019

Keywords:

Blind Spot Detection (BSD)
Hybrid Scheduler
Lane Departure Warning System (LDWS)
Multicore ECU
Task Model

ABSTRACT

In the current approach of Automotive electronic system design, the multicore processors have prevailed to achieve high computing performance at low thermal dissipation. Multicore processors offer functional parallelism that helps in meeting the safety critical requirements of vehicles. The number of ECUs in high-end cars could be reduced by conglomerating more functions into a multicore ECU. AUTOSAR stack has been designed to support the applications developed for multicore ECUs. The real challenges lie in adapting new design methods while developing sophisticated applications with multicore constraints. It is imperative to utilize the most of multicore computational capability towards enhancing the overall performance of ECUs. In this context the scheduling of the real time multitasking software components by the operating system is one of the key issues to be addressed. In this paper, the state of the art scheduling algorithm is reviewed and its merits and limitations are identified. A hybrid scheduler has been proposed, tested and compared with the state of the art algorithm that offers better performance in terms of CPU utilization, average response time and deadline missing rate both in normal and high load conditions.

Corresponding Author:

Geetishree Mishra,
Department of Electronics and Communication Engineering,
BMS College of Engineering,
PO Box no 1908, Bull Temple Road, Bangalore, India.
Email: geetishree@gmail.com

1. INTRODUCTION

The recent trend in automotive industry is to explore all the advanced technologies and utilize those into the automobile to make it an intelligent transport system. The Original Equipment Manufacturers (OEMs) are also constrained by the stringent government regulations for fuel emission and road safety, customers demand for driving comfort, fuel economy, entertainment and connectivity. Increasing complexity, high level of integration requirements and strict regulations for quality standards are all constraints and challenges for automotive software design [1, 2]. In order to incorporate sophisticated features and facilities into the vehicle, more complex and faster computing, multitasking applications are getting developed. ECUs intended to run complex control applications have already been upgraded with the multicore processors. Parallelized application tasks need to be efficiently scheduled and allocated to multiple computing cores to accomplish the functions within the specified deadlines to ensure the system is fail safe. The state of the art task scheduling algorithm for multicore ECUs instructs strict partitioning and mapping of the tasks to definite cores [3]. Tasks are scheduled for the intended cores based on static priority scheme. It works fine in normal running condition of the vehicle upto 65% CPU utilization. In contingency scenarios, when engine speed drastically increases or multiple critical events are sensed, the scheduling of regular applications tasks get affected which leads to missing of deadlines [4, 5]. In order to address this issue, a hybrid scheduling algorithm is proposed in this paper. This proposed algorithm is tested with many task sets representative of applications for multicore engine control unit. The performance of this algorithm is compared with the state of the art algorithm based on standard parameters such as CPU utilization, average response time and deadline missing rate.

The rest of the paper is organized as follows. Section 2 discusses about the scheduling challenges on multicore ECUs, section 3 presents a review on the state of the art algorithm, section 4 presents the proposed hybrid scheduler, performance evaluation and analysis of results are discussed in sections 5 and 6 and the work is concluded in section 7.

2. SCHEDULING CHALLENGES ON MULTICORE ECUS

Task scheduling is a functionality of real-time operating system that essentially refers to determining the sequence in which the various tasks are to be executed adhering to all the timing constraints. A large number of algorithms for scheduling real-time tasks have so far been developed. Scheduling can be either priority driven with static or dynamic priority or it can be event driven like on occurrence of interrupts. The real time operating system scheduler either can have non-preemptive or preemptive or cooperative scheduling mechanism [6]. When the computing unit is a single core chip, the task scheduling algorithms are quite simple and well tested. But the scheduling complexity increases with the multicore implementation.When the number of computing cores is more, the scheduling challenges and constraints are also increased. If the cores are identical in features, any task can be scheduled on any core. But if the core features are different, there will be allocation constraint which may affect the task response time [7]. Also if tasks on different cores have dependency, it incurs high communication cost. If tasks scheduled on different cores have common data sharing then inter core communication will add complexity to the eco system which is difficult to handle in a critical real time automotive system [8].

In multicore ECUs, the task allocation to various cores can be static or dynamic depending on the criticality of the applications. When the tasks are allocated, there should be proper utilization of the cores and workload should be balanced across the cores. Accordingly, different scheduling approaches are available for muticore processors. Some of them are: utilization balancing algorithm, next fit for RMA, bin packing for EDF, myopic algorithm, fault tolerant algorithm etc [9]. Also there are three scheduling approaches such as: global scheduling, partitioned scheduling and clustered scheduling.

Parallel multithreaded software for multicore ECUs requires an optimal scheduling policy to ensure a highly stabilized and temporally deterministic system. According to Automotive Open System Architecture (AUTOSAR) 4.0, there are certain limitations on multicore software implementation. The scheduling algorithms strictly partition the tasks based on similar functionality or periodicity or dependency and assign those tasks to fixed cores [10, 11]. Tasks are scheduled in the intended core based on their statically assigned priority. In the current scenario, tricore microcontroller implemented in the multicore ECU is utilized upto 60% in normal running condition and few cores are even less utilized [11]. When load increases with the speed of the vehicle, it is transferred to under loaded cores.

3. STATE OF THE ART ALGORITHM

In the context of task scheduling for multicore automotive ECUs, the AUTOSAR suggests partitioned static priority scheduling to get predictable response for safety critical applications [1]. It is considered as the state of the art algorithm for task scheduling. Since there is no fixed partitioning method, the problem of partitioning the task threads and allocating those on multiple identical cores have been addressed through various research works. A low-complexity heuristic algorithm to partition the thread sets to a least loaded core and build sequencer table to execute at a least loaded time slot has been proposed [2]. It is observed that, minimum average load of the total load is distributed to each core to achieve a balanced load with task scheduling. In normal running condition of the vehicle it gives feasible schedule with utilization upto 60%. But this algorithm has limitations with the load conditions. As the partitioned tasks are scheduled in their intended core according to the static priority, at heavy load conditions low priority tasks miss their deadlines. Even though the task threads are clustered considering the precedence and collocation constraints and allocated to one core, it is difficult to partition a cluster of independent thread to the core where other threads of the same task have been allocated [12]. In the real scenario, such an independent thread might lose its sequence which may lead to data corruption. So instead of partitioning the task threads, tasks themselves can be partitioned within which the sequence of threads execution could be defined.

4. PROPOSED HYBRID SCHEDULER

In order to address the above identified problems, various task models have been tested with different heuristic algorithms. The scheduling algorithm used for multicore processor environment is P|rj, deadline, precedence constraints |Cmax. This algorithm is designed for solving the α|β|γ=P|rj, prec,~dj|Cmax problem. Here α=P means the target processor has P number of cores, the constraint parameters considered for β factor are release time, deadline and precedence constraints. The algorithm uses modified List Scheduling algorithm to determine an upper bound of the criterion γ=Cmax i,e maximum completion time [13]. From the resulted schedule, it is inferred that, all these mentioned constraint parameters have to be considered while deriving a feasible schedule for real time tasks. Considering slack of the tasks as the criterion for assigning priority, an efficient hybrid scheduler has been proposed in this paper. The task with minimum slack has highest priority. This algorithm has the features of both global and partitioned scheduling and tasks are allowed to migrate from one core to other with a probability of meeting their deadlines. This proposed algorithm is rigorously tested with multiple task models using a Linux kernel based simulation tool for real time multiprocessor scheduling. It is also tested with task sets of high utilization representing the heavy load conditions in contingency scenarios. Its performance has been compared with the state of the art algorithm. The proposed algorithm has six distinct functions: Initialization, Task Distribution, Scheduler, Core Allocation, and Preemption & Migration.

4.1. Initialization

The algorithm starts with the initialization function. All the global list, local list, local variables and flags are initialized. Once the tasks get activated or unblocked, their slack times are calculated. In this work, since periodic sensor tasks characteristics are used to generate the task models, slack is the difference between the period and the execution time at the time of release. The tasks are then sorted in ascending order of their slack at the global queue (1).

$$\text{Slack}=(\text{deadline- release time- execution time})\Rightarrow(\text{period-execution time}) \tag{1}$$

4.2. Task distribution

The sorted tasks are distributed to the local queues of CPU cores. The criterion for distribution is the density of tasks arrival, i,e the number of tasks n in the sorted global list. If n is an even number, then first 0.5n is passed to the local queue1, next 0.25n to local queue2 and remaining 0.25n to local queue3 as the algorithm was tested on tricore processor model. If n is an odd number, then [0.5(n+1)] is passed to the local queue 1, [n- 0.5(n+1)]/2 to local queue 2 and remaining tasks to local queue 3. In this simple strategy of tasks distribution, the consequence is, local queue1 has always more number of tasks than local queue 2 & 3. So the possible migration of tasks is always from queue1 to queue2 or queue3 and not in the reverse direction which considerably reduces the complexity and cost of migration.

4.3. Scheduler

Once the tasks are joined to the corresponding local queues, on SysTick(), the scheduling window i,e time quantum Qt and the laxity of each task is calculated as per equation 2 and the tasks are sorted in ascending order of their laxity. The task with least laxity is assigned with highest priority and scheduled for execution from the head of the queue. The task at the head of the queue is allocated to the core for execution and runs for a time quantum. It continues execution to complete its Worst Case Execution Time (WCET) unless it is preempted by a higher priority task. A task can also migrate to other local queues when there is possibility of missing deadline (2).

$$\text{Laxity=slack- waiting time}\Rightarrow[\text{slack- (present time- release time- }T_{\text{over}})] \tag{2}$$

4.4. Core allocation

In this function, it is checked if the running task is same as the task at the head of the queue. If true, the execution time of the running task is counted and compared with its WCET. If (ET. Trunning=WCET), the task is removed from the core and local queue. Its Execution Time (ET) is reset to zero. But if the condition is false, then the running task is preempted and the scheduled higher priority task goes through the above said process. When a task is blocked on a resource, it is removed from the global queue and joined the wait list. Similarly, when it is terminated on completion of one instance, it is removed from the local list.

4.5. Preemption

When it is identified that, the running task is not same as the newly scheduled task, the preemption function is called. In this function, the Remaining Execution Time (RET) of the task is calculated and compared with the slack of the scheduled task. If (RET. Trunning < Slack. Tqhead), no preemption takes place. Otherwise the running task is deallocated from the core and added to the tail of the local queue and the scheduled task is allocated to the core for execution. The local queue is resorted after an event of preemption.

4.6. Migration

At every event of new tasks arrival at local queues, the migration function is called. In this function laxity of the task at the tail of the queue1 is compared with sum of remaining execution times of all tasks listed before it. If

$$\left(Laxity.T_{LQ1_{tail}} < \sum_{i=1}^{k-1} \mathrm{RET(Ti)}\right) \tag{3}$$

& its precedence task is not added in the queue, migration of the task is allowed.
If $\left(Laxity.T_{LQ1_{tail}} < Laxity.T_{LQ2_{head}}\right)$, the task is migrated to LQ2 else in the same condition, the task is migrated to LQ3. After migration, sorting happens according to the ascending order of laxity and the migrated task gets on to the CPU core immediately and finishes execution within deadline. In the case when both the said conditions are not satisfied, the cumulative RET of both LQ2 and LQ3 are calculated for half of their lengths and compared with the laxity of task at the tail of LQ1. If

$$\left(Laxity.T_{LQ1_{tail}} < \sum_{i=1}^{l/2} \mathrm{RET(Ti)}\right) \tag{4}$$

task is migrated to LQ2 else if

$$\left(Laxity.T_{LQ1_{tail}} < \sum_{i=1}^{\frac{m}{2}} \mathrm{RET(Ti)}\right) \tag{5}$$

task is migrated to LQ3. Here k, l, m are the queue lengths of LQ1, LQ2 and LQ3 respectively.

5. TASK MODEL DESCRIPTION

In this work, the partition static and the proposed hybrid scheduler were run and tested using various task models those represent the applications software for engine control ECU that is designed with three identical computing cores [14]. In engine control unit, two types of tasks are executed: the asynchronous or time-triggered tasks, which are activated periodically and the synchronous or engine-triggered tasks, which are activated at a specific angular position of the engine crank shaft. As a consequence, the frequency of engine-triggered task arrival varies with the speed of the engine. Additionally, the execution time of some of the time triggered tasks may also depend on the speed of the engine. With all these considerations, task models were created representing the real time behavior of the engine. The task characteristics, assumptions considered and Schedulability conditions for the proposed hybrid scheduler are presented here.

The ith task is characterized by a three tuple Ti=(Ci, Ri, Pi). Quantities Ci, Ri, and Pi correspond to the worst case execution time (WCET), the releasing instant and the period. Subsequent instances of the tasks are released periodically. Slack is calculated for each task and is denoted by Si. Si=Pi-WCET-Wt; where Wt= waiting time of a task=(present time- release time- execution time). Table 1 shows few periodic tasks of engine control unit [1] that are used as the model tasks for simulation purpose. Task models are the representation of run time behaviour of the functional codes designed with the consideration of all the timing constraints.

For this scheduler, a set of assumptions were made for the task models, considering the safety critical applications.
- Each task is periodically executed strictly. The initial releasing instant of a task can be chosen freely within its period with the objective of balancing the CPU load over a scheduling cycle [D].
- Periods are chosen as 5ms, 10ms, 20 ms, 50 ms and 100ms for realistic automotive applications.
- The WCET of tasks are randomly chosen to have a CPU utilization between 0.1 to 0.2 assuming that, maximum 5 to 10 tasks will be waiting in the queue for execution at the same instant.
- All cores are identical with regard to their processing speed and hardware features. There are no dependencies between tasks allocated on different cores [15, 16].
- Tasks are free to migrate across the cores when they don't have precedence constraints.

The schedulability conditions are:
- The total CPU utilization.
  $\mu = \sum_{i=1}^{n} \mathrm{Ci/Pi}$, where n=number of tasks
- $\forall n, \mu \leq m$; where m=number of CPU cores
- Slack of any task 'Ti' should be an integer multiple of its WCET 'Ci'.

- Slack Si=Pi-WCET-Wt, where Wt=waiting time.
- Remaining slack of a task in a local queue should be greater than the sum of remaining-execution time of tasks arranged before it.

Table1. Periodic tasks of engine control unit

Periodic Tasks	Description
OS_1ms _Tasks	Engine speed monitoring task at lower RPM.
	Watchdog timer handling task.
OS_2ms _Tasks	Crank sensor signal plausibility check task.
OS_10ms _Tasks	Cam sensor signal acquisition task.
	CAN communication check
OS_20ms _Tasks	Fan control
	Exhaust valve sensor control
	Fuel injector pressure detection.
OS_50ms _Tasks	Oil pump pressure check
	Cam shaft shift checking
OS_100ms _Tasks	Crankshaft signal diagnosis
	Camshaft signal diagnosis.

6. ANALYSIS FOR REAL TIME PERFORMANCE

The Gantt charts of partition static priority scheduling for three periodic task models are shown in Figure 1 to Figure 3. The task models used for simulation had tasks with 5ms, 10ms, 20ms and 50ms periods. Due to strict partitioning of the tasks, those were with 5ms and 10ms periods allocated to core1 and tasks with 20 ms and 50ms period were allocated to core2. Priorities are assigned based on activation rate. It can be clearly observed from core1 Gantt charts of all three task models that, only higher priority tasks have been scheduled initially and medium priority tasks were scheduled with high response time. The low priority tasks had missed deadlines. As less number of tasks was allocated to core2, all had met their deadlines and core utilization was very less. Since no partition for core3, it was completely idle and utilization of the core was zero.

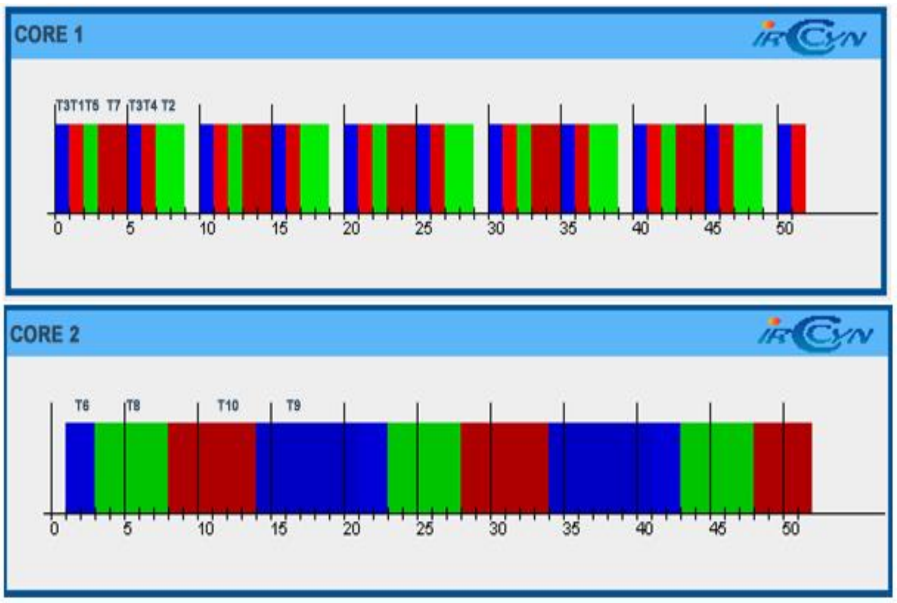

Figure 1. Gantt charts for partition static priority algorithm on model 1

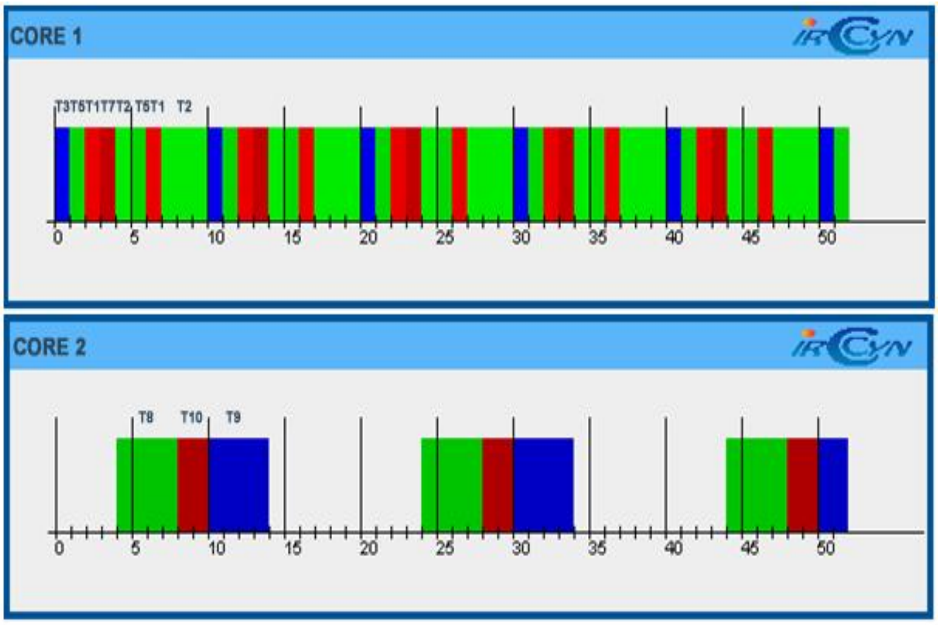

Figure 2. Gantt charts for partition static priority algorithm on model 2

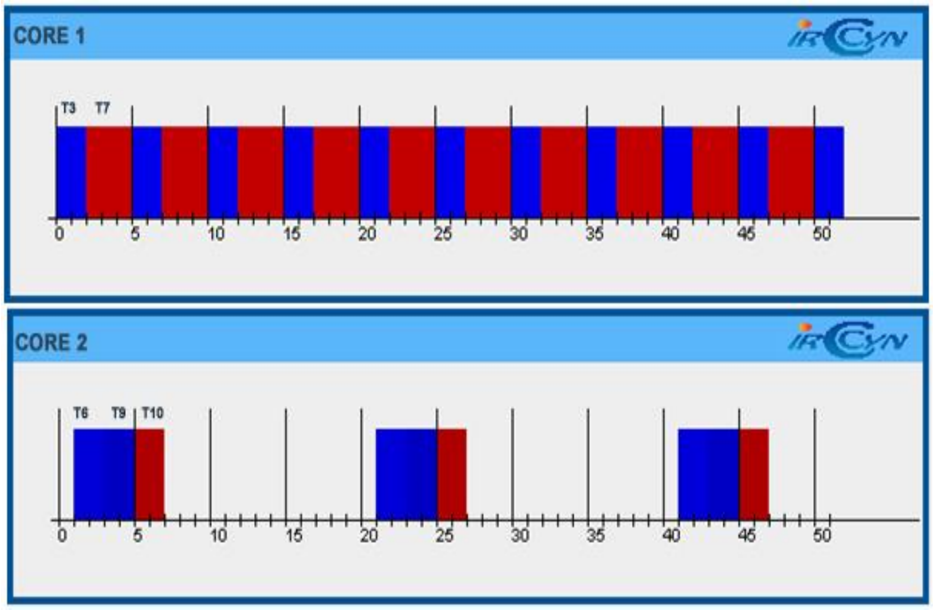

Figure 3. Gantt charts for partition static priority algorithm on model 3

Hybrid scheduler

The result Gantt charts for the same three periodic task models using the proposed hybrid scheduling algorithm are shown in Figure 4 to Figure 6. At every task releasing instant, the task distribution logic passed atleast 50% of the tasks to local queue1 and as a result, it got more populated. With the help of its migration logic, the scheduler moved the tasks at the tail of the queue to other queues with the probability of meeting deadlines. As per the schedulability conditions of the algorithm, it can give a feasible schedule if, $\mu \leq m$. Since the considered task models are relaxed in this respect, the hybrid scheduler had given feasible schedules for all the tasks and also some idle time slots appeared on core2 and core3 Gantt charts.

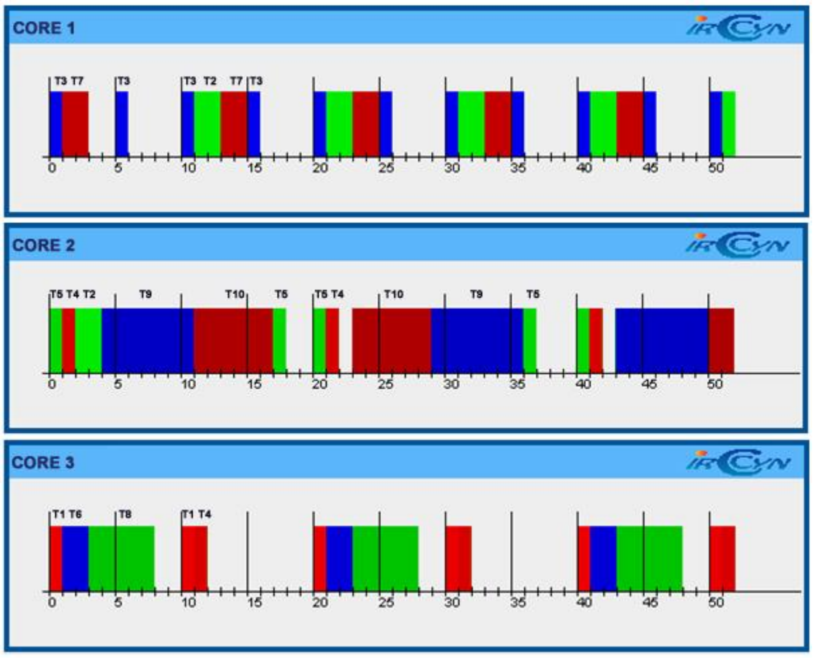

Figure 4. Gantt charts for hybrid scheduling algorithm on model 1

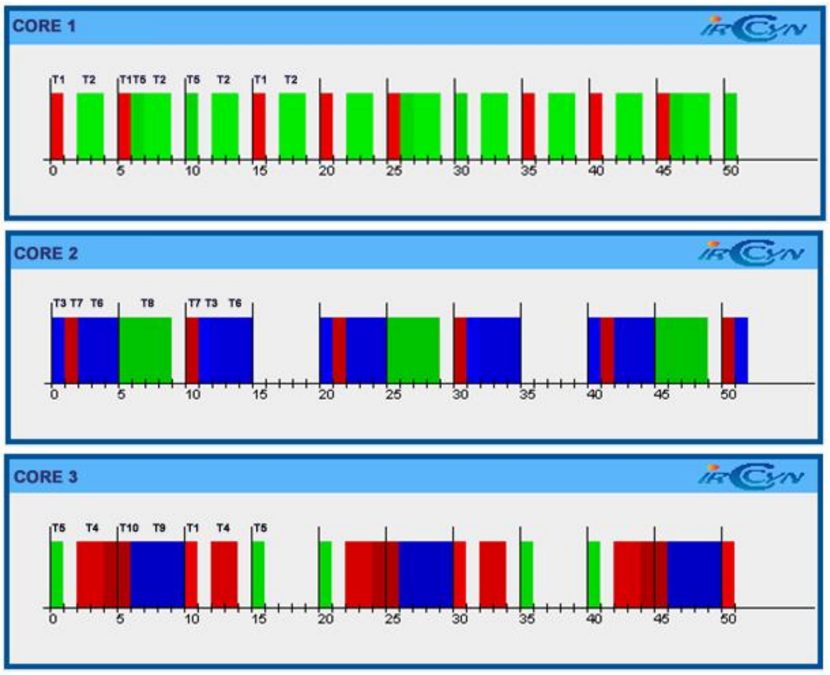

Figure 5. Gantt charts for hybrid scheduling algorithm on model 2

Figure 7 shows the graphical presentation of partition static scheduler and hybrid scheduler depicting the variation of CPU core utilization with the number of tasks after running the algorithms with twenty different task models with varied utilization and number of tasks in each core. It can be observed from Figure 8 that, there is a considerable improvement in response time of tasks scheduled by hybrid algorithm over the partitioned algorithm. It is observed from Figure 9 that, as the average utilization of a CPU core exceeds 65% for a task model, for partition static priority scheduling, lower priority tasks miss their deadlines or do not get scheduled.

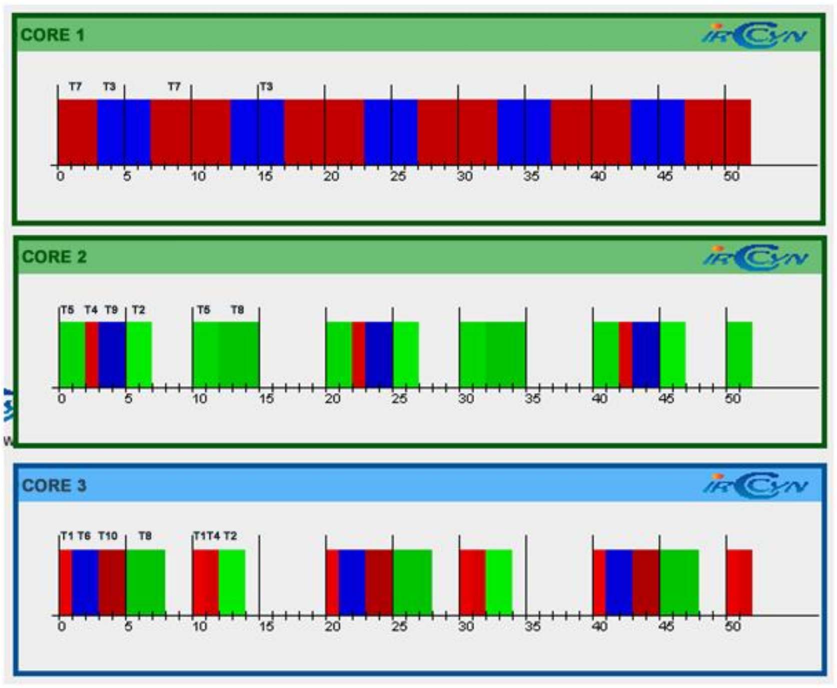

Figure 6. Gantt charts for hybrid scheduling algorithm on model 3

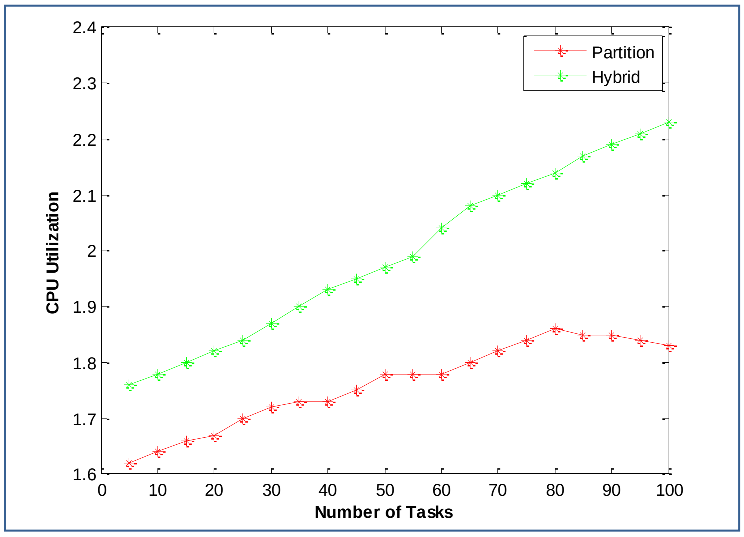

Figure 7. Variation of CPU utilization with number of tasks

In this work, the parameters considered for comparing the performance of proposed hybrid scheduler with the existing partition static priority scheduler are:
The percentage of CPU core utilization μ is given by

$$\mu = \sum_{i=1}^{n} Ci/Pi$$

Average response time for the scheduled tasks $= \frac{(\sum_{i=1}^{n} Rt)}{n}$

Where Response time Rt=Execution time+Waiting Time [17, 18] and deadline missing rate which is the percentage of total tasks scheduled that miss their deadlines

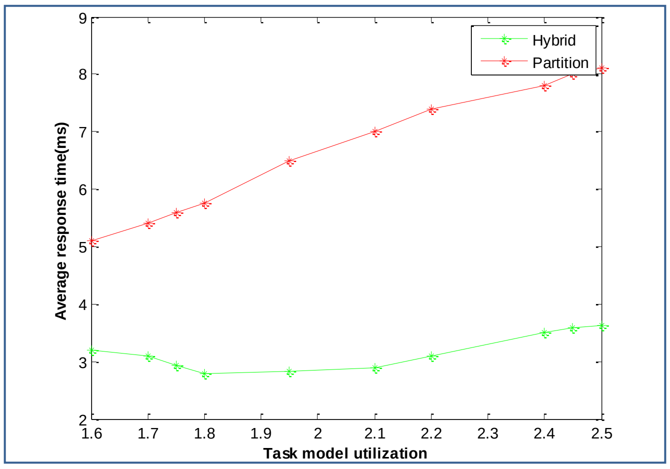

Figure 8. Variation of average response time with task utilization

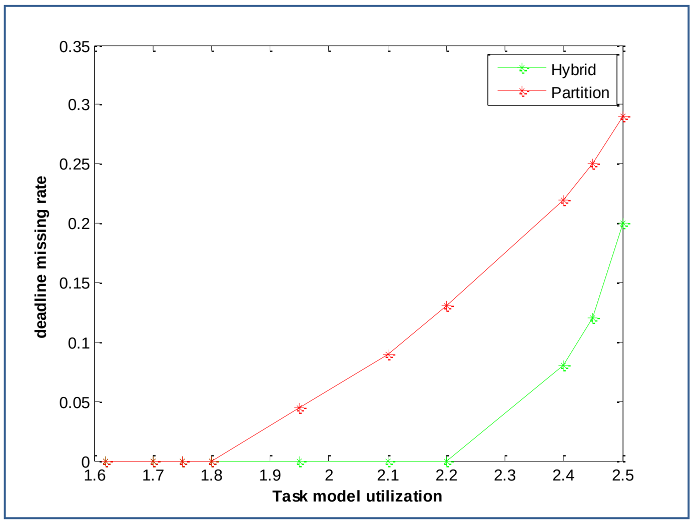

Figure 9. Variation of deadline missing rate with task utilization

7. CONCLUSION

The main focus of the proposed task scheduling algorithm was to improve the core utilization and reduce the deadline missing rate. In this work, both the proposed hybrid and the state of the art partition static algorithms had been rigorously tested for twenty different task models of varied task characteristics, representatives of engine control ECU functionalities. It has been verified that, the proposed algorithm has considerable improvements over the existing partition static priority algorithm based on the performance parameters such as CPU core utilization, average response time of tasks and deadline missing rate. The main motive behind developing this hybrid scheduling algorithm was to distribute the tasks among the available cores instead of strict partitioning according to the task periods. In this proposed algorithm, tasks were allowed to migrate from one queue to another, hence able to reduce the response time and meet the deadlines. As all the cores share the workload, higher utilization of the cores has been achieved when the work load increased.

REFERENCES

[1] Aurélien Monot, Nicolas Navet, Françoise Simonot, Bernard Bavoux, "Multicore Scheduling in Automotive ECUs", *Embedded Real Time Software and Systems-ERTSS 2010.*

[2] Giorgio Buttazzo, Enrico Bini, Yifan Wu, "Partitioning Real Time Applications Over Multicore Reservations" *IEEE Transactions on Industrial Informatics*, pp 302–315, March 2011.

[3] Samarjit Chakraborty, Marco Di Natale, Heiko Falk "Timing and Schedulability Analysis for Distributed Automotive Control Applications", pp 349-350, *Proceedings of ninth ACM International Conference on Embedded Software 2011*.
[4] Nicolas Navet, Aurélien Monot, Bernard Bavoux, Françoise Simonot-Lion, "Multi-Source and Multicore Automotive ECUs–OS Protection Mechanisms and Scheduling", *IEEE International Symposium on Industrial Electronics*, pp 3734-3741, 2010.
[5] Transportation Research Board, "The Safety Promise and Challenge of Automotive Electronics", *National Research Council of the National Academies*, 2012.
[6] Anssi, S. Tucci-Piergiovanni, S. Kuntz, S. Gerard, and F. Terrier. "Enabling Scheduling Analysis for AUTOSAR systems", *14th IEEE International Symposium on Object/Component/Service-Oriented Real Time Distributed Computing*, pp 152-159, March 2011.
[7] Jinkyu Lee, Insik Shin, "Demand Based Schedulability Analysis for Real Time Multicore Scheduling", *The Journal of Systems and Software*, Vol. 89, pp 99-108, October 2013.
[8] Karthik Lakshmanan, Gaurav Bhatia, Ragunathan (Raj) Rajkumar, "AUTOSAR Extensions for Predictable Task Synchronization in Multi-Core ECUs", *SAE International 2011*.
[9] Mike Holenderski, Reinder J. Bril and Johan J. Lukkien, "An Efficient Hierarchical Scheduling Framework for the Automotive Domain", *Real-Time Systems, Architecture, Scheduling, and Application*, pp 67-94, 2012.
[10] Geetishree Mishra, K S Gurumurthy, "Dynamic Task Scheduling on Multicore Automotive ECUs", *International Journal of VLSI design and Communication Systems*, Vol 5, No 6, pp 1-8, December 2014.
[11] Wenfeng Shena, Zhaokai Luo, Daming Wei,Weimin Xu, Xin Zhu, "Load Prediction Scheduling Algorithm for Computer Simulation of Electrocardiogram in Hybrid Environments", *The Journal of Systems and Software*, Vol. 102, pp 182-191, January 2015.
[12] D Claraz, F Grimal, T Leydier, R Mader, G Wirrer, "Introducing Multicore at Automotive Engine Systems", *Springer 5th International Chassis Symposium*, 2014.
[13] Arunachalam Annamalai, Rance Rodrigues, Israel Koren, Sandip Kundu "Dynamic Thread Scheduling in Asymmetric Multicores to Maximize Performance-per- Watt", *IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, pp 964-971, 2012.
[14] Sangeet Saha, Arnab Sarkar, Amlan Chakrabarti, "Scheduling Dynamic Hard Real-Time Task Sets on Fully and Partially Reconfigurable Platforms", *IEEE Embedded Systems Letters*, pp 23-26, 2015.
[15] Hamid Arabnejad, Jorge G. Barbosa, "List Scheduling Algorithm for Heterogeneous Systems by an Optimistic Cost Table", *IEEE Transactions on Parallel and Distributed Systems*, pp 682-694, 2014.
[16] Björn Andersson, Gurulingesh Raravi, "Real Time Scheduling with Resource Sharing On Heterogeneous Multiprocessors", *Springer Real Time Systems*, pp 270-314, 2014.
[17] Abusayeed Saifullah, David Ferry, Jing Li, Kunal Agrawal, Chenyang Lu, and Christopher D. Gill, "Parallel Real-Time Scheduling of DAGs", *IEEE Transactions on Parallel and Distributed Systems*, pp 3242–3252, 2014.
[18] Giovani Gracioli, Antônio Augusto Fröhlich, Rodolfo Pellizzoni, Sebastian Fischmeister, "Implementation and Evaluation of Global and Partitioned Scheduling in a Real-Time OS", *Springer Real-Time Systems*, pp 669-714, 2013.

BIOGRAPHIES OF AUTHORS

Dr Geetishree Mishra received PhD from Visvesvarya Technological University, Belagavi in the field of Embedded Systems and MTech in Digital Communication from BMS College of Engineering, Bangalore. She is currently working as Assistant Professor, Department of Electronics & Communication Engineering, BMS College of Engineering, Bangalore, India. Her research interests include Embedded Systems, Real time Systems, Architecture and Operating Systems.

Dr Rajeshwari Hegde received PhD from Bangalore University and Master of Engineering in Electronics from BMS College of Engineering, Bangalore and Bachelor of Engineering in Electronics and Communication Engineering from National Institute of Engineering, Mysore, She is currently working as Associate Professor, Department of Telecommunication Engineering, BMS College of Engineering, Bangalore, India. Her research interests include Embedded Systems and Wireless Communication.