International Journal of Computer Applications (0975 – 8887)
Volume 117 – No. 17, May 2015

A Mixed Approach Scheduling Algorithm for Multicore Automotive ECUs at Contingency Conditions

Geetishree Mishra
BMS College of Engineering, Bangalore, India,

K S Gurumurthy
Reva Institute of Technology, Bangalore, India

ABSTRACT

Automotive ECUs have been upgraded with multicore processor implementation. It has the benefits of achieving high computing power without increasing the clock speed. System developers partition the automotive application tasks to have parallelizability and avoid interference between various software modules. Task intensive applications are assigned to multiple CPU cores. To improve the performance of such systems, there has to be an efficient task scheduler. In this regard, the Automotive Open System Architecture (AUTOSAR) suggests partitioned static priority scheduling for parallelized software for the multicore ECUs. In this approach, the difficulty lies with task clustering and partitioning for specific cores. There is no exact criterion to be followed to partition the tasks. Due to which cores are not balanced with loads. Under contingency conditions, there are chances of tasks missing deadlines. This paper addresses this issue by exploring a mixed approach scheduling algorithm which has features of both static and dynamic scheduling and also few adaptations of partitioned and global scheduling. With this algorithm, high load conditions under contingency consequences are tested. This algorithm was run and tested using a scheduling simulator with real time task models of periodic tasks, angle synchronous interrupts and event triggered interrupts. The performance parameters considered here are, the % of core utilization, response time, deadlines missing rate. It has been verified that, this proposed algorithm is able to find a feasible schedule under various contingency scenarios and contributes to improve the safety level of the vehicle.

Key Words

ECU, Multicore, RTOS, Scheduling, OEM, AUTOSAR.

1. INTRODUCTION

Current trends in automotive industry have directed towards introducing more sophisticated features and innovations in vehicle electronics and software. Therefore demand on computational resources is growing rapidly. In automotive domain, multicore processors have been integrated into electronic control units (ECUs) to run complex algorithms for task intensive applications [1, 3]. Complex applications have specific requirements on software and hardware. Automotive Open System Architecture (AUTOSAR) is a standard to manage the increasing complexity and defines a methodology for efficient development of such parallelized systems [2]. Multicore implementation provides high level of software integration, parallel computing and provides the safety requirements of the system. Multicore improves the performance of computationally intensive algorithms by executing multiple independent threads of same application parallely on different cores. Many different applications can also be integrated into the same ECU sharing common hardware resources. Safety critical functions can run on multiple cores to provide a fault tolerant system. To optimize the performance of such systems, there is always a need of an efficient scheduler function of the real time operating system embedded in the ECU. To ensure the deterministic behavior of the safety critical systems, AUTOSAR suggests partitioned static priority scheduling for multicore ECUs [2]. In this approach, a feasible schedule is assured prior to run time for periodic tasks but the identified issues are: partitioning of tasks is the most tedious process and static priority scheduling handles aperiodic and sporadic tasks inefficiently. As a consequence, multicore processors are not utilized to their maximum capabilities and it is difficult to meet the worst case processing requirements [8]. In this paper, this issue has been addressed by developing a mixed approach scheduling algorithm in which dynamic scheduling approach has been explored and few features of both partitioned and global scheduling are utilized. This algorithm is tested at different contingency conditions when multiple unexpected events occur through hardware interrupts and are feasibly scheduled along with the regular periodic tasks. This paper is organized as follows: task model representative of time critical automotive applications is stated in section 2. In section 3, the current scenario of task scheduling in automotive domain is presented. In section 4, the hybrid scheduling algorithm is explained. In section 5, the evaluation of the algorithm performance is presented through simulation studies. Section 6 presents the concluding remarks.

2. TASK MODEL

In this paper, we have considered application tasks of a tri core engine control ECU to be scheduled at normal running conditions of the vehicle, considering many sequential processes run within these tasks. In automotive ECUs two types of tasks are normally executed: the asynchronous or time-triggered tasks, which are activated periodically by the system tick and the synchronous or engine-triggered tasks, which are invoked at the engine crank teeth position [4]. As a result, the frequency of occurrence of engine-triggered task varies with the speed of the engine. In this paper, the proposed scheduler is tested with different task models considering the real time behavior of the engine. The ith task is represented 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 of the task [5,6,7]. For all the periodic tasks, deadline is equal to the period. Slack of a task is the maximum amount of time it can be delayed for execution before meeting its deadline. Slack is denoted by Si. Si= Pi – RET; where RET= remaining execution time of a task. Aperiodic tasks included in the task model to verify the performance at contingency conditions.

3. EXISTING TASK SCHEDULING METHOD

Applications with deterministic realtime requirements, such as for automotives mission critical systems, impose severe constraints on the design and implementation of real-time system software and its functions. Automotive safety critical applications such as anti lock braking system (ABS), electric power steering (EPS), electronic stability program (ESP), traction control system, collision avoidance system or engine control unit must function reliably with the temperature or pressure swing within the vehicle or even when exposed to harsh environments [4]. The real time system software should have failsafe features to handle such situations. Scheduling of safety critical application tasks is an important functionality of RTOS, majorly contributes to achieve real time requirements. AUTOSAR suggests partitioned static priority scheduling to achieve time deterministic behavior of safety critical applications [2,3]. It is the distributed scheduling method used in automotive domain wherein, tasks are partitioned and scheduled separately for individual core. In this method, related tasks or interrupt service routines are clustered and allocated to a defined core. For example, the basic software and complex device drivers are partitioned for one core and engine crank teeth angle synchronous tasks are for another core [4]. But this partitioning of tasks being a tedious process and there is no availability of optimum task partitioning scheme, computing cores are underutilized. Also static priority algorithms though provide feasible schedule for periodic tasks of hard deadlines with low run time overhead, they are inefficient to schedule at heavy load conditions and also aperiodic tasks whose characteristics are not known prior to run time. Even though multicore processor offers parallelizability in the hardware, the computing cores are not optimally utilized. These are the motivating factors for further research on task scheduling for multicore automotive ECUs. These issues have been addressed in this paper by developing a mixed approach scheduling algorithm and verifying its performance in various contingency conditions.

3.1 Basic Multicore System

In Fig.1, the tricore system has separate level 1 caches for each core, but share a common level 2 cache, memory controller, interrupt subsystem and I/O subsystem. The implementation of multiple cores incurs greater design complexity both for hardware and software. Applications running on different cores need to have efficient interprocess communication (IPC) mechanisms, shared-memory data structures, appropriate synchronization mechanism and primitives for shared resources protection [9].

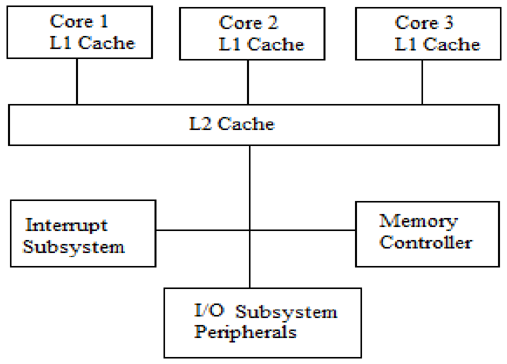

Fig.1 A generic multicore system

There are three multiprocessing modes: offered by multicore processors: symmetric, asymmetric and partitioned. In automotive domain, partitioned approach is used. By which, an application is strictly bound to one core and cannot leverage to others even if they are idle. This approach is adopted looking into the safety criticalities of automotive applications. In automotive domain, AUTOSAR is the driving architecture for multicore software design. The partition multiprocessing mode is suggested by AUTOSAR and supported by the system software. The task scheduling function of system software needs to be optimized to enhance the performance of multicore ECUs. As all the computing cores are not utilized to their maximum capabilities, there should be an adaptive method of scheduling considering the interprocess dependency so that, utilization of cores is improved and at the same time safety criticalities of the applications are not affected [10].

3.2 Partition static priority scheduling

Partition Static Priority Scheduling Algorithm

```
1   Initialize global queue and local queue1, local queue2 & local queue3.
2   On tasks activation:
3   Add tasks to global queue
4   For Ti; where i = 1 to n; n= no. of tasks in global queue at ‘t’
5           Partition tasks(Pi) {
              (i) Group tasks of periods within a fixed bound into task clusters.
             (ii) Identify a core for a cluster.
            (iii) Allocate the task clusters that have a locality constraint
                  to the corresponding queue.
             (iv) Map the task clusters to the identified local queues.
            {add to local queue1,2 & 3}    // Pi= period of the task
10  For Li; where i = 1 to n; n= no. of queues
11      Sort (tasks)          // descending order of priority
    Pr1>Pr2>…..PrN      // Pr =  assigned priority
12      Allocate CPU to the task with Pr1.
13  If the running task is not the task at the queue_head,{Preempt the running task}
14  If  interrupt occurs,
        (i)       Map the interrupt to a core.
        (ii)      Schedule as per sorted priority.
```

In the partition static priority scheduler, there are three main functions. One is to partition the tasks into task clusters and map to the predefined core. Second one is to sort the tasks in descending order of their assigned priority and allocate CPU to the highest priority task for execution. Third one is preempt the running task if required. Regular crank teeth synchronous tasks are invoked through interrupts and interrupts are basically assigned with higher priority. They are also pre-partitioned and scheduled along with periodic tasks following the same mechanism. In any contingency condition, when multiple events occur and invoke simultaneous interrupts of relatively higher assigned priority, the running tasks are mostly preempted to cater to these interrupts [11]. As a result of which, some of the low priority tasks miss their deadlines.

4. A NOVEL MIXED APPROACH SCHEDULING ALGORITHM

In the proposed mixed approach scheduling algorithm, the three main logic used are for task distribution, task pre-emption and task migration. To get feasible schedule for more number of tasks, migration from one core to other is allowed. This reduces the number of pre-emption also. As pre-emption incurs more overhead than migration, inclusion of migration logic is an advantage of the proposed algorithm. The tasks after being admitted to the global queue get distributed to the three local queues passing through the distribution logic. Slack, the difference between period and WCET of a task is the utilised parameter here. As it is the maximum duration any task can be delayed without missing the deadline, it is exploited in this algorithm to make a sequence of execution, to find out the possibility of missing deadline being in a queue and get migrated or pre-empt a running task. The task with least remaining slack is always scheduled first [12]. The crank teeth synchronous tasks get invoked by interrupts but have periodicity. So they are scheduled along with other periodic tasks. In contingency conditions when multiple interrupts come simultaneously, their criticalities are checked and tasks are sorted according to descending order of their criticalities. If number of interrupts are more than the number of cores, pre-empt all the running tasks and get the high severity interrupts executed. If the checked criticalities are within the set threshold, they can be scheduled passing through the slack sorting process.

Mixed approach Scheduling Algorithm Pseudocode

```
1    Initialize global queue and local queue1, local queue2 & local queue3.
2    On task activation:
3      Globalqueue.Add(task)
4        For Ti; where i = 1 to n;   n= no. of tasks in global queue
         { Compute(slack);}
         Sort(slack);      // ascending order of slack at globalqueue.

7       Distribute(tasks)           // To local queues
      As n/2,( (n – (n/2))/2 and remaining to local queue1, 2 and 3 sequentially.

8        For Li; where i= 1 to n     //n=3, no.of local queues.
9        {Sort(remaining slack )}          // in ascending order at localqueue
10            If(!Task(queuehead) = Task.running)
                {Preempt(tasks)}

12            If (Sn <  ∑RET (Tj)); j varies from 1 to n-1    // RET = remaining execution time
                 {Migrate (tasks)}                        // Sn= remaining slack of 'nth' task.

13   If(Interrupt){
        {Calculate (load)                   // at each queue.
     Localqueue.Add(Interrupt)              // with least load
     If(criticality>threshold)
     {Preempt(running task)}
     Calculate(slack)                       // remaining slack.
     Sort(queue)                            // ascending order of remaining slack
     Schedule(least slack)}
14   If number of interrupts  > m at 't'    // m=no. of cores.
     {sort(criticality) }                   // descending order of assigned criticality
     Preempt(all running tasks)
     Schedule(interrupts)                   // 'm' interrupts with high criticality
```

5. RESULTS AND DISCUSSIONS

In this paper, the proposed mixed approach scheduling algorithm is compared with the existing approach of partition static priority scheduling algorithm. A tricore processor and three different task models representative of engine control unit functionalities were used in simulation. Each model has ten numbers of periodic tasks denoted as T1 to T10. To represent the contingency scenario, five aperiodic tasks are included which are denoted as A1 to A5. The performance parameters are: core utilization, average response time and missed deadline. To run and validate the algorithms, a multiprocessor scheduling simulator is used that gives a Gantt chart for each core as the result.

The Gantt charts for task model M1, M2 and M3 are shown in fig.2. In every model, numberings 1 & 2 are for results of partition static scheduler and 3, 4 & 5 are for results of mixed approach scheduler. In every task model, due to strict partitioning, no task is going to Core 3 with partition static scheduler. The aperiodic tasks used in every model to test the contingency condition have deadlines which come under the partition of Core 1. As a result, these aperiodic tasks having higher priority than other periodic tasks are scheduled first pre-empting the running task in core 1. Since all are scheduled for execution in one core, the consequence is deadline missing of many tasks. So to get better results in these critical situations, the task partitioning method should be modified to be more flexible else there will be hazardous failure of the system. In the mixed approach scheduling, tasks are distributed to all the cores depending on the number of tasks released. All the cores are utilized both for periodic and aperiodic tasks. So core utilization is improved with better response time of the tasks and there is no missing of deadlines. As the result shows in fig.2, for partition static scheduler, core 1 & 2 are highly loaded always and core 3 is idle and remains in sleep mode and it wakes up only when loads get transferred to it from other cores in heavy load conditions. There is an upper bound of total work load to have feasible schedule for both the algorithms discussed in section 6.

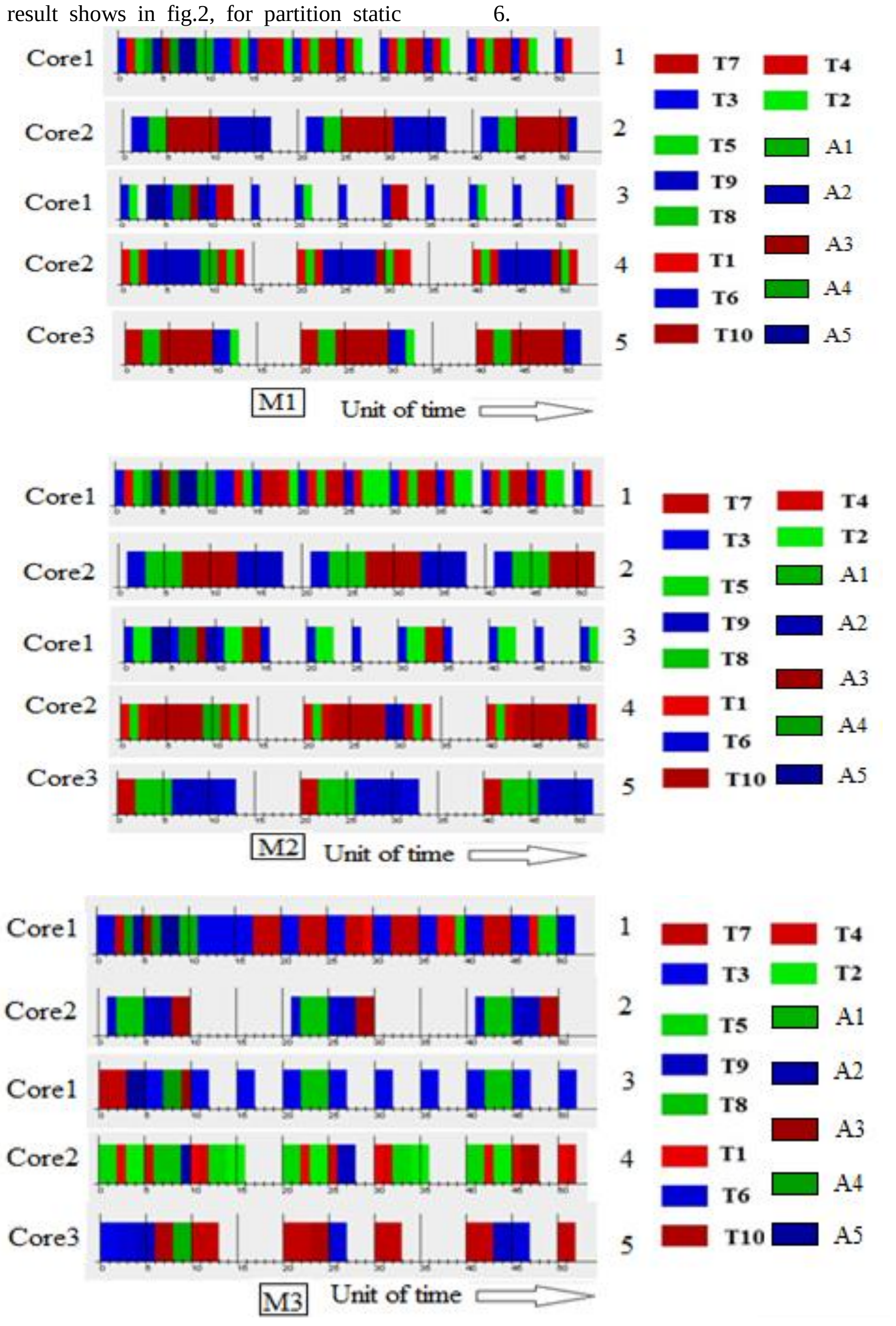

Fig.2 Gantt charts for model 1, 2 & 3

6. EVALUATION OF MIXED APPROACH ALGORITHM IN CONTINGENCY CONDITIONS

For performance evaluation of the proposed mixed approach algorithm compared to existing partition static priority algorithm, three parameters are considered in this paper: CPU utilization, average response time and deadline missing rate.

6.1 CPU utilization

The total CPU utilization required by a task model is $U = \sum_{i=1}^{n} Ci/Pi$, Where 'n'= no. of tasks [1,3,4]For task model 1, utilization U is 2.3, for task model 2, U is 2.5 and for task model 3, it is 2.7 considering both periodic and aperiodic tasks in contingency conditions. Each task model has run with both the partition and mixed approach scheduling algorithms. Table 1 gives the utilization percentage of each core for each task model tested on both the algorithms. Utilization of each core is calculated for the simulation duration. As it is clearly shown in the table 1, for partition static priority scheduling, there is load imbalance across the three cores and in mixed approach scheduler, all the three cores are utilized as a result of task migration and work load distribution across the cores.

6.2 Average response time

Response time of a task is: (Rt) = ET+WT, where ET= execution time and WT= waiting time from its release till completion in any occurring instance. Average response time is the sum of response times of all scheduled tasks divided by the number of tasks scheduled, that is: $\text{Avg Rt} = \sum_{i=1}^{n} \text{Rt}/\text{n}$, Where n = no. of tasks scheduled [3]. Table.2 shows the comparison for both the algorithms. Comparisons are shown separately for periodic and aperiodic tasks. It is clearly observed from the comparison that, in mixed approach scheduler, all the periodic tasks are feasibly scheduled with less average response time along with the critical aperiodic tasks whereas in partition static scheduler low priority tasks are preempted for which average response time for periodic tasks is delayed by the scheduling duration of aperiodic tasks.

6.3 Deadline missing rate

Deadline of periodic tasks is assumed as equal to its period and it is defined as a task parameter for aperiodic tasks. For each of the scheduling algorithms, with the imposition of contingency conditions, the number of periodic tasks missing their deadlines on every task model within the simulation duration was observed. It was observed that, as the workload demand exceeds 60% for a task model, for partitioned static priority scheduling (PSPS), there is exponential missing rate of deadlines. Another important aspect is, in PSPS, the priorities are assigned based on rate monotonic approach (RMA) and the worst case bound in RMA is: □< n($2^{1/n}$-1) [12], where □ is the utilization of the task set and ‘n’ is the number of tasks in the task set. As for simulation, ten number of tasks are considered in the task models, the upper bound of utilization □ is 71%. In contingency conditions, when high criticality higher priority tasks are invoked, only those higher priority tasks meet their deadlines those have utilization within this bound. For all the three task models, the average utilization of the tasks should be restricted to the range of 0.1≤ [1/n(∑ Ci/Pi)] ≤ 0.2 where n is the number of tasks under test and ‘i’ varies from 1 to n. Experiments show that, mixed approach scheduling (MAS) algorithm provides a feasible schedule for the task models with 80% utilization of each core at any given time and deadline missing rate is 20-40% of the waiting task as the utilization goes beyond 80%.

Table.1 Comparison of CPU utilization

CPU Utilization						
	Partition Static Priority Scheduler			Hybrid Scheduler		
Task Model	Core1	Core2	Core3	Core1	Core2	Core3
M1	88%	84%	0	46%	74%	72%
M2	96%	82%	0	60%	70%	72%
M3	100%	56%	0	64%	76%	66%

Table. 2 Comparison of average response time

Average Response Time (Avg.Rt)				
	Partition Static Priority Scheduler		Mixed approach Scheduler	
Task Models	PTask	ATask	PTask	ATask
M1	9.2ms	4ms	2.5ms	5ms
M2	10.1ms	4ms	4ms	5ms
M3	15ms/8tasks	4ms	3.7ms	5.2ms

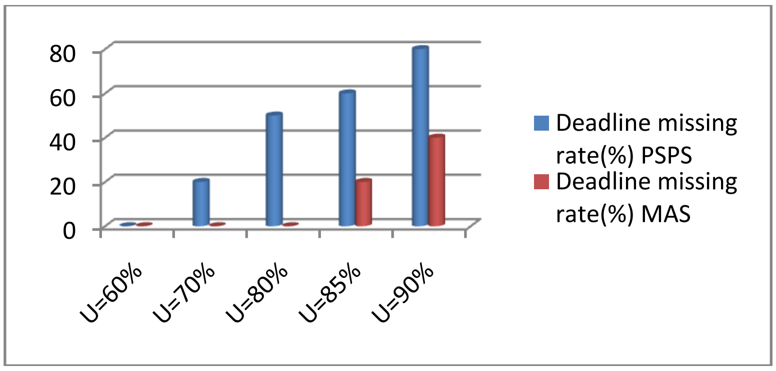

Fig.2 Comparison of deadline missing rate between PSPS & MAS algorithms.

7. CONCLUSION

In multicore automotive ECUs, the currently used task scheduling algorithm is partition static priority scheduling where the tasks are strictly partitioned into clusters based on their periods before execution in a selected core. As a result of which, CPU cores are not utilized to their maximum capabilities in normal running conditions [3,8]. In this paper, a mixed approach scheduling algorithm is proposed for multicore automotive ECUs. Both the algorithms have been tested for three periodic task models representatives of Engine Control ECU functionalities. For verifying the performance of these algorithms at various contingency conditions, aperiodic tasks have been introduced in the task models. It has been verified that, this proposed algorithm has considerable improvements over the existing partitioned static priority scheduler based on the performance parameters such as: CPU core utilization, average response time of tasks and deadline missing rate. In the proposed algorithm, tasks are distributed among the cores to utilize the availability and are allowed to migrate from one queue to another. The average response time of tasks is reduced and all the tasks meet their deadlines. In this algorithm, all the cores share the total workload at any scheduling instant so higher utilization is achieved with increase in work load. With these performance improvements, this proposed algorithm has comparatively more migration overhead. At contingency severity consequences, the migration of time critical tasks should be bounded to ensure meeting the deadlines which could be considered as future work on this algorithm.

8. REFERENCES

[1] ”Multicore Scheduling in Automotive ECUs” By Aurelien Monot, Nicolas Navet, Francoise Simonot, Bernard Bavoux, Embedded Real Time Software and Systems- ERTSS May 2010.

[2] AUTOSAR version 4.2.1, www.autosar.org

[3] “Multi-source and multicore automotive ecus: Os protection mechanisms and scheduling” By N. Navet, A. Monot, B. Bavoux, and F. Simonot-Lion.. In Proc.of IEEE Int’l Symposium on Industrial Electronics, Jul. 2010.

[4] “A Case Study in Embedded Systems Design: An Engine Control Unit” , By Tullio Cuatto, Claudio Passerone, Claudio Sanso E, Francesco Gregoretti, Attila JuresKa, Alberto Sangiovanni, Design Automation for Embedded Systems, Vol, 6, 2000.

[5] “Dynamic Scheduling for Emergency Tasks on Distributed Imaging Satellites with Task Merging.” By Jianjiang Wang; Xiaomin Zhu; Dishan Qiu; Yang L T, Parallel and Distributed Systems, IEEE Transactions, June 2013.

[6] “Harmonic Aware Multicore Scheduling for fixed priority real time systems.” By Ming Fan; Geng Quan, Parallel and Distributed Systems, IEEE Transactions. March 2013.

[7] “Improvement of Real time Multicore Schedulability with forced Non Preemption.” By Jinkyu Lee; Shin K G , Parallel and Distributed Systems, IEEE Transactions. Jan 2014.

[8] “Dynamic Scheduling Strategies for Avionics Mission Computing” By David L. Levine, Christopher D. Gill and Douglas C. Schmidt, Proceedings of Digital Avionics Systems Conference, 17th IEEE, Nov 1998.

[9] “Dynamic Task Scheduling on Multicore Automotive ECUs”By Geetishree Mishra, K S Gurumurthy, International Journal of VLSI design & Communication Systems (VLSICS), DOI: 10.5121/vlsic.2014.5601, Vol.5, No.6, December 2014.

[10] “Task Scheduling of Real-time Systems on Multi-Core Architectures” By Pengliu Tan, 2009 Second International Symposium on Electronic Commerce and Security IEEE, DOI 10.1109/ISECS.2009.161.

[11] ”Demand-based Schedulability Analysis for Real Time Multicore Scheduling” By Jinkyu Lee, Insik Shin, Journal of Systems and Software ELSEVIER October 2013.

[12] “Load-prediction Scheduling Algorithm for Computer Simulation of Electrocardiogram in Hybrid Environments” By Wenfeng Shen, Zhaokai Luo, Daming Wei, Weimin Xu, Xin Zhu, Journal of Systems and Software ELSEVIER January 2015