DYNAMIC TASK SCHEDULING ON MULTICORE AUTOMOTIVE ECUS

Geetishree Mishra[1], K S Gurumurthy[2]

[1]BMS College of Engineering, Bangalore, India,
[2]Reva Intitute of Technology, Bangalore, India

ABSTRACT

Automobile manufacturers are controlled by stringent govt. regulations for safety and fuel emissions and motivated towards adding more advanced features and sophisticated applications to the existing electronic system. Ever increasing customer's demands for high level of comfort also necessitate providing even more sophistication in vehicle electronics system. All these, directly make the vehicle software system more complex and computationally more intensive. In turn, this demands very high computational capability of the microprocessor used in electronic control unit (ECU). In this regard, multicore processors have already been implemented in some of the task rigorous ECUs like, power train, image processing and infotainment. To achieve greater performance from these multicore processors, parallelized ECU software needs to be efficiently scheduled by the underlaying operating system for execution to utilize all the computational cores to the maximum extent possible and meet the real time constraint. In this paper, we propose a dynamic task scheduler for multicore engine control ECU that provides maximum CPU utilization, minimized preemption overhead, minimum average waiting time and all the tasks meet their real time deadlines while compared to the static priority scheduling suggested by Automotive Open Systems Architecture (AUTOSAR).

KEYWORDS

OEM, ECU, Multicore, Scheduling, AUTOSAR.

1. INTRODUCTION

Electronic Control Units (ECUs) fulfill the objectives and requirements of a modern automobile which is designed to be safer, more comfortable and fuel efficient. Therefore the number of ECUs has been increased continuously over the years. Advanced functionalities demand higher computational capabilities from ECUs. Multicore processors have emerged to be the current processing unit not only for high-end servers but also for embedded control systems [10, 11]. Multicore processor features with parallel processing, compact chip size and lower power consumption. Performance is improved, the amount of processes per core is reduced and better reliability of the on-chip communication is assured by the multicore implementation. In the current scenario, OEMs are moving towards multicore to exploit parallelism, to accommodate more functions on one ECU and distribute them across the computational cores [1, 3]. In effect, some of the task intensive ECUs catering to telematics, infotainment and power train are already upgraded with multicore processors. Performance optimization of such multicore ECUs can be

DOI : 10.5121/vlsic.2014.5601

achieved by utilizing parallelism effectively for which, numerous runnables should be efficiently scheduled for each core [3,4]. The operating system should have an efficient scheduling mechanism to schedule the tasks without corrupting the shared data and without leaving any CPU core idle at any running instant [5]. Without an optimal schedule, there can be considerable variations in average task response times and logical correctness of the results [6]. Such anomalies can cause instabilities in the physical system leading to performance degradation and even safety hazards. In this paper, a hybrid dynamic scheduler is proposed for the most task intensive engine control ECU.

2. CASE STUDY - ENGINE CONTROL UNIT

In this work, engine control unit is chosen as the target ECU for task scheduling, as it has maximum number tasks to be executed both periodic and event driven and it has already been implemented with multicore. Engine control unit is usually connected to a large cluster of sensors like, intake air temperature sensor, engine coolant temperature sensor, intake manifold absolute pressure (MAP) sensor, mass air flow sensor, throttle position sensor, crankshaft speed sensor, camshaft sensor and knock sensor. It is connected to various actuators like idle speed motor, electronic throttle body, fuel pressure regulator and fuel injector and delivery control. Basically those are stepper motors and solenoids directly connected to the ECU [7]. For each sensor, there has to be a task to receive the sensory signal after being conditioned, a task to process it as required according to the control algorithm and another task to send appropriate signal to the intended actuator. Each task has a large number of threads to be executed sequentially with data dependency. Independent threads are executed in parallel by multiple cores. For each engine ECU functionality like, air charge management, engine cooling management, battery management, air fuel management or on-board diagnostics, a large number of inputs to be received, parameters and local variables to be managed and many outputs are expected to be delivered. Because of high level of interaction between the control functions, the shared data are required to be protected efficiently [4, 6]. Three different task schedulers used are, synchronous scheduler, asynchronous scheduler and background scheduler [7]. Synchronous scheduler is used for tasks that need to be executed at certain crank teeth. Asynchronous scheduler also called the time-base scheduler is executed just after the power up scheduler initialization. Background tasks run, when the CPU is idle basically they are busy-wait loops [12, 13]. Besides that, various interrupt service routines are executed to respond to hardware events. Right from the engine start till stop of the vehicle, engine ECU runs with its software of several 1000 lines of code.

3. AUTOSAR ON MULTICORE SCHEDULING

According to AUTOSAR 4.0, there are certain limitations on multicore software implementation.

- The scheduling algorithms strictly assign tasks statically to cores to ensure deterministic response for the real time critical tasks.

- The resource algorithm is not supported across cores. Resources can be shared between tasks that are allocated to the same core but not among tasks/ISRs which are bound to different cores.

OSEK counter and auto started alarms can be used to implement task activation mechanism. The task synchronization issue is addressed by implementing schedule table. The Autosar schedule

table shown in Figure.1. has certain defined set of expiry points. Each expiry point has certain tasks to be activated, settings to be done for some events and offset from the start of the schedule table [8]. OSEK counter drives the iteration over these expiry points on the schedule table. So one tick on the counter corresponds to one tick on the schedule table. Constraints apply to the delays between adjacent expiry points and the delay to the logical end of the schedule table. According to Autosar, the hard real time safety critical tasks are scheduled statically to computational cores as per the fixed priority assigned. In this static priority scheduling approach, the tasks have to be partitioned well with the order of criticality and an appropriate priority assignment scheme has to be adopted to assign priorities prior to run time, which is always a tedious job. The computational cores are sometimes underutilized and the lower priority tasks often miss their deadline though not the most safety critical ones. In this paper, a global dynamic priority approach has been explored.

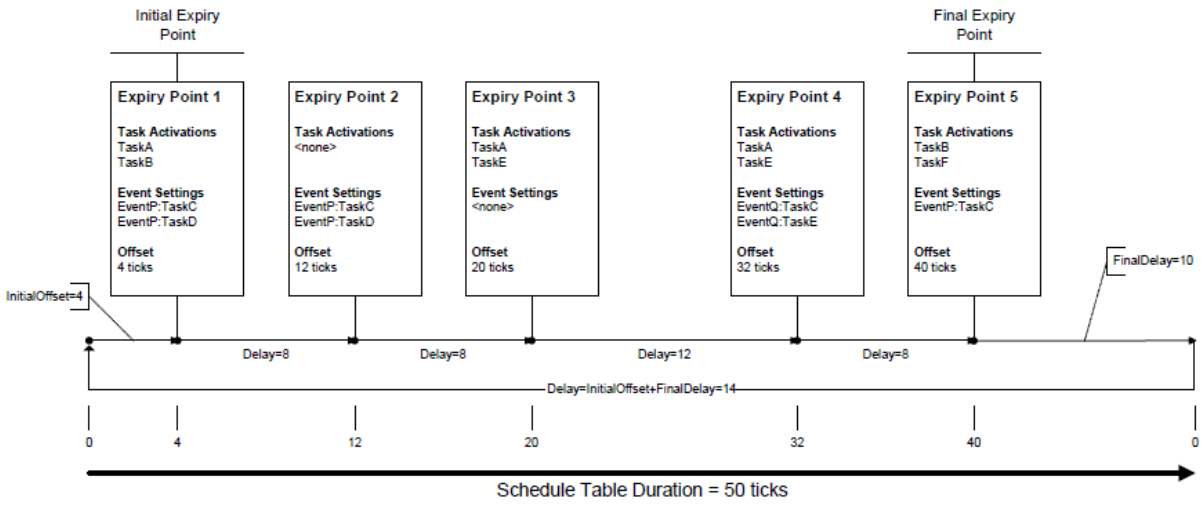

Figure.1. Schedule table for AUTOSAR ECU

4. THE PROPOSED SCHEDULER MODEL

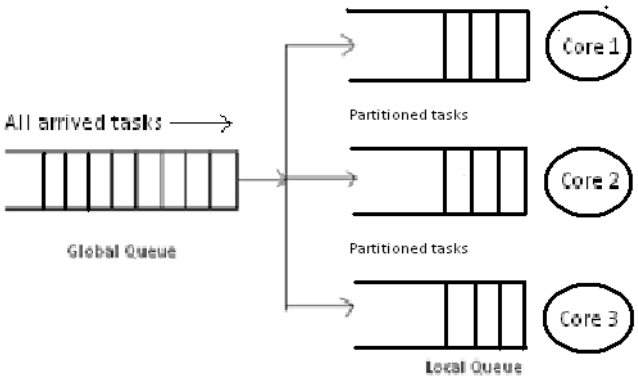

Figure.2. Scheduler model

The scheduler model shown in above Fig.2. has a combination of both global and partitioned queue architecture. All the tasks arrive at the global queue, where the slack is calculated for each and priorities are assigned dynamically. The task with least slack gets highest priority. Then the three tasks pass on to three partitioned queues to utilize all the available cores. When a new task arrives at the partitioned queue in the presence of previous task, a comparison happens between their slack and tasks are arranged in ascending order of their slack in the queue with an effort to meet the deadlines. In this simulation setup, a task set of 10 number of tasks are considered for an asynchronous or timebase scheduler. The task attributes for the ith task are: {releasing instant ri, WCET wi and period pi }, where WCET is the worst case execution time. Precedence constraint is imposed on some of the tasks based on dependency. A tri-core processor implementation is considered for these tasks to be scheduled for. The scheduled tasks are released for execution by the dispatcher task which is characterized by a dispatching table of definite duration.

5. SCHEDULING PROBLEM

The scheduling problems are categorized by notation (α|β|γ) proposed by Graham and Blazewicz [14,15]. This notation consists of three parts. The first part 'α' describes the processor environment, the second part 'β' describes the task characteristics and constraints of the scheduling problem and the last part 'γ' denotes the optimality criterion. In this simulation set up, for a three processor environment, α=3. Since all the tasks are release time constrained and they are periodic, period is the implied deadline for each instance and precedence constraint is also imposed on some of the tasks, β=|release time, precedence & deadline constraint| and γ= Cmax, the maximum completion time to achieve an optimal schedule.

6. WORKING OF THE PROPOSED MINIMUM SLACK FIRST ALGORITHM

At a new arriving instant at global queue,

- Calculate the slack of each task arrived .Where, Slack= Period-WCET.
- Sort the tasks in increasing order of their slack i,e S1< S2 <S3…….Sn.
- Assign priorities to these tasks dynamically by giving highest priority to the task with least slack.
- If precedence constraint imposed on the task, Pass the task to a partitioned queue preferably to the one where its precedence task has been allocated.
- If no precedence constraint, pass that task to a local queue based on early availability of that CPU core.
- If :
  - the no. of tasks arrived at the global queue at a particular instant is > the no. of CPU cores,
  - Compare the total WCET of all the tasks at each partitioned queue.
  - Assign the first task in the remaining list at global queue to the core that has least WCET.
- Else:
  - wait for the next new arriving instant.

  At the local partitioned queue,
- If: new task arrives in the presence of previous task,
  - If the precedence task of the new arrival is the last task waiting in the queue, no comparison required.

- Else
  - Compare the slack of the new task with the WCET of the previous task.
  - If: $S_{\text{new arrival}} < WCET_{\text{previous task}} + \text{Remaining } WCET_{\text{running task}}$ & $S_{\text{previous task}} > WCET_{\text{new arrival}}$,
    - Swap the waiting position of these tasks in the queue.
  - Else if : $S_{\text{new arrival}} < WCET_{\text{previous task}} + \text{Remaining } WCET_{\text{running task}}$ & $S_{\text{previous task}} < WCET_{\text{new arrival}}$,
    - Get the $task_{new}$ to migrate to another local queue whose CPU core is expected to be available early, getting information from the global queue.

7. TASK ATTRIBUTES TABLE

Table.1. Task attributes table

Tasks	Ri	WCET	Period	Precedence constraints	Static Priority	Slack	Dynamic priority assigned	1st instance
T1	0	1	4	-	2	3	3	0-1
T2	1	2	4	T1	2	2	2	2-4
T3	0	2	5	-	3	3	4	1-3
T4	1	1	3	T3	1	2	1	3-4
T5	0	2	3	-	1	1	1	0-2
T6	1	2	8	T5	5	6	3	3-4
T7	0	3	5	-	3	2	2	0-3
T8	2	3	7	T1,T7	4	4	1	5-8
T9	3	2	10	T6,T8	6	8	2	9-11
T10	3	2	7	T5	4	6	1	4-6

The task attributes table given in above table. 1 has release time Ri, worst case execution time WCET, period and static priorities assigned for each of the ten tasks considered [2, 9]. These are the parameters used for static priority scheduling. In addition to these parameters, the precedence constraints are considered for dynamic priority scheduling. The slack is the calculated parameter based on which dynamic priorities are assigned. For example, at 0^{th} instant, four number of tasks i,e T1, T3, T5 and T7 are released. So there could be only four levels of priorities that can be assigned. These priorities are assigned at the global queue to dispatch the tasks to the local partitioned queues based on their corresponding core's early availabilities. The first instance of all the tasks according to minimum slack based dynamic scheduling is shown in the last column of the attributes table.

8. SEQUENCER TABLE AS PER STATIC PRIORITY

	Time Units										
Core 1	T5	T5	T7	T7	T1	T7	T5	T5	T8	T5	T5
Core 2	T1	T4	T3	T5	T5	T3	T3	T7	T7	T7	T4
Core 3	T3	T2	T2	T8	T4	T2	T2	T4	T1	T2	T2
	0	1	2	3	4	5	6	7	8	9	10

Figure.2. Sequencer table

Fig.2 shows a sequencer table for the tasks to be executed in three computational cores in an expected sequence as it is given. At every single unit of time the scheduler has to take the decision in favor of three highest priority tasks among the ready list of tasks arrived as well as tasks with remaining execution time. In the priority list, the smallest number indicates highest priority for the most frequent task.

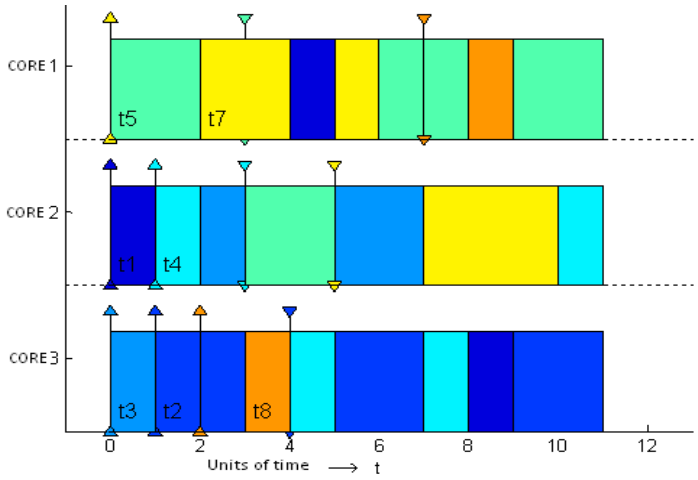

Figure. 3. Simulation result

9. IMPLEMENTATION OF DYNAMIC SCHEDULER AND RESULTS

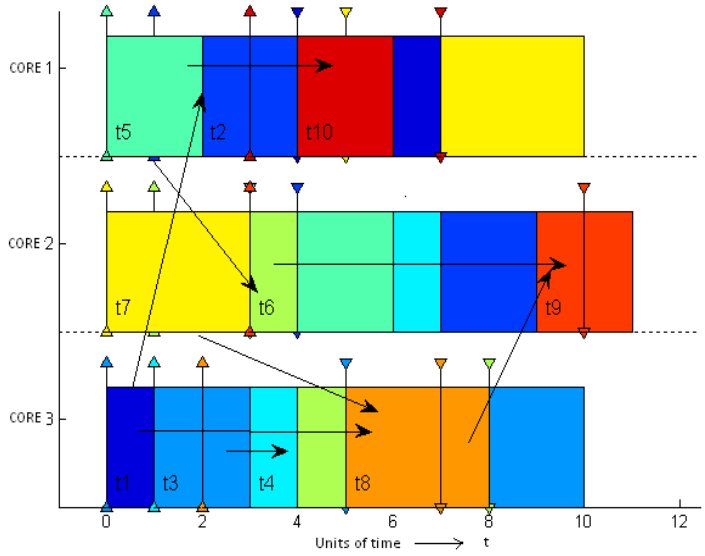

Figure. 4. Simulation result

The simulation results are shown in the above fig. 3 and fig. 4. These result graphs are Gantt charts of ten number of tasks being scheduled for three computational cores. Fig. 3 shows the Gantt chart results of the simulation of static priority scheduling. In this simulation, few noticeable features are there. Tasks T6, T9 and T10 being assigned with least priorities, are not

getting scheduled and have missed their deadlines even for the first instance. T3 has migrated from core3 to core2 to complete its WCET after being preempted by T2. Based on the parameter attributes and constraints imposed on the tasks, both the static and the dynamic algorithms have run through the global and partitioned queues one after the other. Fig. 4 shows the Gantt chart results of the simulation of dynamic priority based minimum slack first scheduling. At every arrival instant at the global queue, slack for each task is calculated and dynamic priorities are assigned. For this dynamic priority scheduling, precedence constraints have been imposed on certain tasks. Tasks with the precedence constraints are mostly collocated at a CPU core. At every arrival instant at the partitioned queue, the slack of the new task is compared with the total WCET of previous tasks and the currently running task. Either the tasks are sorted in ascending order of their slack and scheduled for execution or tasks have been preempted and migrated to other available core in case there is a probability of missing the deadlines. In the result Gantt chart, T6 has migrated from core2 to core3. All the tasks have satisfied their precedence constraints and all have met with their deadlines. None of the CPU cores is idle at any point of time. So 100% CPU utilization is achieved within the simulation duration. To minimize the preemption overhead, the tasks are allowed to continue their execution unless there is a probability of missing the deadline. Table 2 given below shows the number of periodic instances of each of the 10 tasks within the simulation duration for both the static and dynamic scheduling and Table 3 shows the comparison of performance parameters based on the results out of both static and dynamic scheduling run for the given task set.

Table 2: Tasks with no of instances

Scheduling Methods	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
Static	3	3	2	4	4	0	2	1	0	0
Dynamic	2	2	2	2	2	1	2	1	1	1

Table 3: parameter comparison

Scheduling Methods	CPU Utilization	Deadline Miss	Average Response Time	Preemption	Migration
Static	100 % By higher priority tasks.	30% for 1st instance	4 units	30%	20%
Dynamic	100% By fair allocation	3% for 3rd instance	1.6 units	10%	10%

10. CONCLUSION

In this paper, a dynamic task scheduling algorithm has been proposed for multicore Engine control ECU of Automotive electronic system. Engine control ECU being a safety critical, hard real time system and highly task intensive, the stringent requirement is, all the tasks have to meet their deadlines. Fixed static priority and partitioned task scheduling for multicore ECUs have been suggested by AUTOSAR. While adhering to it, there is always a challenge in assigning priorities to the tasks and high level of difficulty in partitioning the tasks by which certain CPU cores remain under utilized [3, 8]. In this work, a model taskset with appropriate time attributes has been tested both with static priority scheduling and the proposed dynamic priority algorithm, where slack is the utilizing parameter. The performance parameters are compared for the results of both the algorithms. It is observed that, in static priority scheduling, CPUs are highly utilized

by higher priority tasks only and the lowest priority tasks are consistently missing their deadlines while in dynamic priority case, there is a significant improvement in terms of missing deadlines and there is a fair allocation across all priorities. Average response time, number of pre-emption and migration have been improved considerably. Since slack is considered to assign dynamic priorities and to calculate the relative deadlines, unless there is a probability of missing the deadline, the task does not get preempted or migrated to other core. So the context switching overhead is also reduced.

REFERENCES

[1] Yu Hua, Lei Rao, Xue Liu, Dan Feng "Co-operative and Efficient Real-Time Scheduling for Automotive Communications". IEEE international conference on Distributed Computing Systems 2014; ISBN-978-1-4799-5168-0; DOI- 10.1109/ICDCS.2014.22.

[2] Aurélien Monot, Nicolas Navet, Bernard Bavoux, and Françoise Simonot-Lion, "Multisource Software on Multicore Automotive ECUs—Combining Runnable Sequencing", IEEE Transactions on Industrial Electronics, Vol. 59, No. 10, pp 3934-3942, October 2012

[3] Rajeshwari Hegde, Geetishree Mishra, K S Gurumurthy, "Software and Hardware Challenges in Automotive Embedded Systems", International Journal of VLSI design and Communication Systems (VLSICS) Vol.2, No.3, pp 165-174, Sep 2011 DOI: 10.512/VLSIC.2011.2314.

[4] A. Monot, N. Navet, F. Simomot, and B. Bavoux. "Multicore scheduling in automotive ecus". In Proc. of The Embedded Real-Time Software and Systems (ERTS2), May 2010.

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

[6] C. Aussagues, D. Chabrol, V. David, D. Roux, N. Willey, A. Tournadre, and M. Graniou. Pharos, "A multicore os ready for safety-related automotive systems: results and future prospects". In Proc. of The Embedded Real-Time Software and Systems (ERTS2), May 2010.

[7] http://www.bosch motorsport.de/media/msd/ downloads/dokumentation/ steuergeraete_1/ Databook_ Engine_ Control_Units_2014.pdf

[8] AUTOSAR 4.0 specification, available at: www.autosar .org.

[9] Zhang Jie, Yang Fumin, Tu Gang, "A Dynamic Scheduling Model for Real-Time Tasks in Reliable System" 2009 International Conference on Networks Security, Wireless Communications and Trusted Computing.

[10] P. Leteinturier (Infineon Technologies). "Multi-core processors: Driving the evolution of automotive electronics architectures"; http://www.embedded.com/design/multicore/, 2007.

[11] Tao Xie, Andrew Sung, Xiao Qin, "Dynamic Task Scheduling with Security Awareness in Real-Time Systems" Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) 1530-2075, 2005.

[12] Anita Mittal, G.Manimaran, C.Siva Ram Murthy, "Integrated Dynamic Scheduling of Hard and QoS Degradable Real-Time Tasks in Multiprocessor Systems" Journal of Systems Architecture, Vol. 46, Issue-9, Pages-793-807, July 2000.

[13] G. Manimaran, C. Siva Ram Murthy, and K. Ramamritham. A new approach for Scheduling of parallelizable tasks in real-time multiprocessor systems, Real-Time Systems, 15:39-60, July 1998.

[14] RL Graham, EL Lawler, JK Lenstra, A.H.G Rinnooy Kan - "Optimization and Approximation in deterministic sequencing and Scheduling –A Survey" Annalysis of Discrete Mathematics, 5; 1977, Pages 287-326.

[15] J Blazewicz, J K Lenstra, A.H.G Rinnooy Kan, "Scheduling Subject to resourse constraints: classification and complexity" , Dicrete Applied Mathematics V-5; Jan 1983, Pages- 11-24.