International Journal of Scientific and Technical Advancements
ISSN: 2454-1532

Multicore ECU in Automotive Domain with FPP Scheduling

T. Sivamani[1], R. Punitha[2]
[1, 2]Department of ECE, Hindusthan Institute of Technology
Email address: [2]rpunitha20@gmail.com

***Abstract*—**The aim of ECU group is to research the Multicore architecture for automotive safety applications to meet hard real-time embedded systems timing and reliability constraints. The automotive industry needs to change its architectural approach in developing vehicle electronics systems. By integrating more number of functions in a limited set of ECUs. These new features involve greater complexity in the design, development, and verification of the software applications. Hence, automotive industry manufacturers require efficient tools and design methodologies to fulfill their needs in various aspects. In this project, we address the problem of sequencing the infinite number of runnables on a limited set of distinct cores as the sequencer tasks in order to uniforming the CPU load over time.
The paper presents the low-complexity heuristics to partition and build sequencer tasks that execute the runnable set on each core. The scheduling problem is being addressed globally at the ECU level, where other OS tasks are scheduled on the same cores as the sequencer tasks. Further we are reducing the execution time of numerous runnables using intertask communication between distinct Multicore ECU's in efficient manner.

***Keywords*—**Automotive; autosar; load balancing; multicore; runnable; scheduling; static cyclic scheduling.

I. INTRODUCTION

Multisource software running on the same electronic control unit (ECU) is becoming increasingly widespread in the automotive industry. This case is one of the main reasons that car manufacturers want to reduce the number of which grew up above 70 for high-end cars. One major outcome of the Automotive open system architecture (AUTOSAR) initiative and, more specifically, its operating system (OS) is to help car manufacturer's shift from the "one function per ECU" paradigm to more centralized architecture designs by providing appropriate protection mechanisms. Another crucial evolution in the automotive industry is those chips manufacturers are reaching the point where they can no longer cost effectively meet the increasing performance requirements through frequency scaling alone. This condition is one reason that multicore ECUs are gradually introduced in the automotive domain. The higher level of performance provided by multicore architectures may help simplify in-vehicle architectures by executing on multiple cores that the software previously run on multiple ECUs. This possible evolution toward more centralized architectures is also an opportunity for car manufacturers to decrease the number of network connections and buses. As a result, parts of the complexity will be transferred from the electrical/electronic architecture to the hardware and software architecture of the ECUs. However, static cyclic scheduling makes it easy to add functions to an existing ECU.

In practice, important architectural shifts are hindered by the carryover of ECUs and existing subnet works, which are widely used by generalist car manufacturers. The extent to which more centralized architectures will be adopted thus remains unsure. Multicore ECUs are also helpful for other use cases. For example, they bring major improvements for some applications that require high performance such as high-end engine controllers, electric and hybrid powertrains and advanced driver assistance systems, which sometimes involve real time image processing. These multicore platforms also offer additional benefits such as higher level of parallelism, allowing for more segregation, which may help meet the requirements of the International Organization for Standardization (ISO) 26262, which concerns functional safety for road vehicles. Furthermore, in multicore architectures, some core can be dedicated to a specialized usage such as handling low-level services. Now, the challenge is to adapt existing design methods to the new multicore constraints.

The scheduling of the software components is one of the key issues in that regard, and it has tobe revamped. The introduction of multisource and multicore will induce drastic changes in the software architecture of automotive ECUs. Section II introduces the most likely scheduling choices and the literature relevant to the task scheduling in multiprocessor automotive ECUs. Then, Section III presents solutions for the scheduling of numerous software modules when only a few OS tasks are allowed. This paper builds on the study published in, where it was assumed that only one sequencer task was running on each core of the ECU to schedule the runnables. In Section V, we consider how we can build several sequencer tasks while possibly scheduling other tasks on the same core, and we discuss how we can globally analyze the schedulability of such systems. For clarity, "sequencing" refers to the scheduling of runnables, whereas "scheduling" is solely used for tasks.

II. SCHEDULING IN THE AUTOMOTIVE DOMAIN

A. *Scheduling Design Choices for Multicore ECUs*

In this section, we explain and justify, particularly in light of predictability requirements, the multicore scheduling

T. Sivamani and R. Punitha, "Multicore ECU in automotive domain with FPP scheduling," *International Journal of Scientific and Technical Advancements*, Volume 1, Issue 4, pp. 45-49, 2015.

approach, which is, to the best of our knowledge, the most widely considered method in the automotive industry.

1) Partitioning scheduling scheme

In a multicore system, either the tasks are statically allocated to the cores or they can dynamically be distributed at runtime to balance the workload or migrate functions to increase availability. The latter approach involves complex tasks and resource interactions that are difficult to predict and validate. Thus, approaches that rely on static allocation (i.e., partitioning) and deterministic mechanisms such as periodic cyclic scheduling are more likely to be used in the automotive context, and this is the option taken within the AUTOSAR consortium. Scheduling tasks on a multiprocessor systems under the static partitioning approach has been well studied; However, the works we are aware of deal with online algorithms such as fixed priority preemptive (FPP) or earliest deadline first (EDF) and do not consider the static cyclic scheduling of tasks.

2) Static cyclic scheduling

The static cyclic scheduling of elementary software modules or runnables is common, because there are usually many more runnables than the maximum number of tasks allowed by automotive operating systems such as OSEK/VDX or AUTOSAR OS. Thus, runnables must be grouped together and scheduled within a sequencer task (also called a dispatcher task). In this paper, we focus on how we can sequence large runnable sets on multicore platforms using a static partitioning approach. Indeed, the static task partitioning scheme is very likely to be adopted, at least, in a first step, because it is conceptually simple and provides better predictability for ECU designers compared with a global scheduling approach. We aim at developing practical algorithms whose performances can be guaranteed to build the dispatcher tasks on each core and to schedule the runnables within these dispatcher tasks to comply with sampling constraints and, as long as possible, uniformize the CPU load over time. This latter objective is, of course, important to minimize the hardware cost and to facilitate the addition of new functions, as typically done in the incremental design process of car manufacturers. This objective is achieved by desynchronizing the runnable release dates. Precisely, the first release date of each runnable, called its offset, is determined to uniformly spread the CPU demand over time.

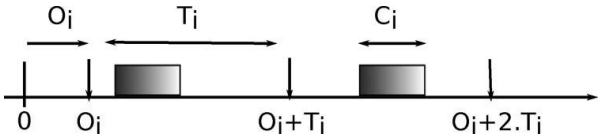

Fig. 1. Model of the runnables.

The configuration algorithms developed in this paper are closely related to (monoprocessor scheduling of tasks with offsets) and (scheduling of frames with offsets), but it is applied to multicore and goes beyond as we provide lower bounds on the performances. Because the problem is of practical interest in the industry, there are in-house tools at the car manufacturers and commercial tools that have been developed for configuring the scheduling, such as Real Time at-Work However, the proprietary algorithms used in these tools can usually not be disclosed, and they are sometimes specialized for some specific usage.

B. Model Description

In this paper, we consider a large set of *n* periodic elementary software modules, also called runnables that will be allocated on an ECU that consists of *m* identical cores. In practice, a runnable can be implemented as a function that is called, whenever appropriate, within the body of an OS task.

1) Runnable characteristics

The *i*th runnable is denoted by *Ri* = (*Ci, Ti,Oi, {R}, Pi*). Quantities *Ci, Ti*, and *Oi* correspond, respectively, to the worst case execution time (WCET), the period (i.e., the exact time between two successive releases), and the offset of *Ri*. The offset of a runnable is the release date of the first instance of that runnable, and subsequent instances are then periodically released. The choice made for the offset values has a direct influence on the repartition of the workload over time. A set of interrunnable dependencies is denoted by *{R}*.Indeed, due to specific design requirements, such as shared variables, some runnables may have to be allocated on the same core, and the set *{R}* is used to capture these constraints. In addition, some specific features, such as input/output (I/O) ports located on a given core, may require a runnable to be allocated onto a specific core. This locality constraint is expressed by *Pi*.

2) Dispatcher task

Runnables are scheduled on their designated core using a dispatcher task or a "sequencer task,"which stores the runnable activation times in a table and releases them at the right points in time. A dispatcher task is characterized by the duration of the dispatch table *Tcycle*, which is executed in a cyclic manner, and by a quantum *Ttic*, which is the duration of a slot in the table. Typically, we may have, for example, *Tcycle* = 1000 ms and *Ttic* = 5 ms. Note that *Tcycle* must be a multiple of the greatest common divisor of the runnable periods and the least common multiple (LCM) of these periods must be a multiple of *Ttic*. As a result, a dispatch table holds *Tcycle/Ttic* slots.

3) Schedulability condition

Assuming that we only consider 6runnable scheduling, the system is schedulable and, thus,can safely be deployed if and only if the following conditions are satisfied on each core.

a) The runnables are strictly periodically executed.
b) The initial offset of each runnable is smaller than its period.
c) The sum of the WCET of the runnables allocated in each slot does not exceed a given threshold, which is typically chosen as the duration of the slot, i.e., *Ttic*.

III. Runnable Sequencing Algorithms for Multicore ECUs

In this section, we present algorithms and, when possible, derive lower bounds on their efficiency to schedule large numbers of runnables on multicore ECUs.

T. Sivamani and R. Punitha, "Multicore ECU in automotive domain with FPP scheduling," *International Journal of Scientific and Technical Advancements*, Volume 1, Issue 4, pp. 45-49, 2015.

International Journal of Scientific and Technical Advancements
ISSN: 2454-1532

Because automotive OSs can only handle a limited amount of OS tasks, the sequencing of runnables has to be done within dispatcher tasks. The first step of the approach is to partition the runnable sets onto different cores. The next and last step is to determine the offsets between the runnables allocated on each core to balance the load over time.

Algorithm 1: Partitioning of the runnable set.
Input: Runnable set *{Ri}*, number of cores *m*

1) Group interdependent runnables into runnable clusters. Independent runnables become clusters of size.
2) Allocate the runnable clusters that have a locality constraint to the corresponding cores.
3) Sort the runnables clusters by decreasing order of CPU utilization rate ρ=∑i(Ci/Ti).
4) Iterate over the sorted clusters.
   a) Find the least loaded (LL) core.
   b) Assign the current cluster to this core.

This presents practical solution for scheduling activities according to both the static cyclic and priority-driven paradigms as it is becoming a need in automotive multicore ECUs and other complex embedded systems with dependability requirements such as in the aerospace domain. Multiprocessor task scheduling is a key research area in high performance computing. without violating the deadlines, a set of software modules called runnables on each processors should be assigned and allocated at run time in order to balancing the load over cpu .To overcome the run time complexities, the runnables are allocated statically (i.e partitioning) on each distinct cores.

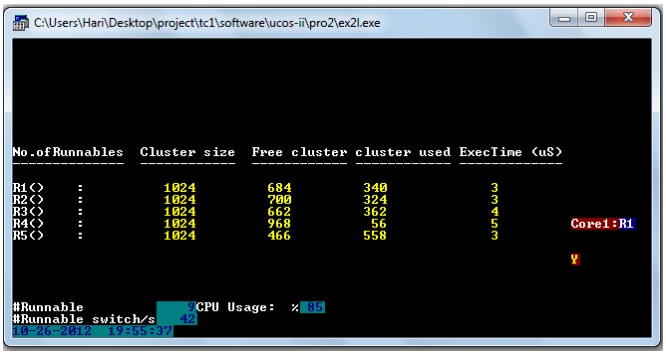

Fig. 2. Simulation result partitioning the runnables.

Algorithm 2: Assigning runnables to slots—LL heuristic.
Input: Runnable set *{Ri}*, *Ttic*, *Tcycle*

1) Sort runnables *Ri* such that *Ttic*≤*T*1≤· · ·≤*Tn*≤*Tcycle*.
2) For *i* = 1. . . *n*
   a) Look for the LL slot in the (*Ti/Ttic*) first slots.
   b) Allocate *Ri* in every (*Ti/Ttic*) slot, starting from this slot.

Step 1 runs in *O*(*n* log *n*). Step 2 iterates *n* times over steps 2a and 2b, which run, respectively, in (*Ti/Ttic*)≤ (*Tcycle/Ttic*) and (*Tcycle/Ti*) ≤ (*Tcycle/Ttic*). As a result, this algorithm runs in *O*(*n*(log *n*+(max*i*{*Ti*}/*Ttic*)+(*Tcycle*/ min*i*{*Ti*}))≤ *O*(*n*(log *n* + 2(*Tcycle/Ttic*)).

The problem of sequencing runnables the LL algorithm proposed by Grenier for the frame offset allocation on a controller area network. The intuition behind the heuristic is simple. The first important criterion is to have the lowest maximum load in the cycle, because this will determine the feasibility of the schedule and the possibility of adding further functions later in the lifetime of the system. The maximum load over all slots is also referred to as the peak load. In the second step, a more fine-grained assessment of the uniformity of the load balancing can be given by the standard deviation of the load distribution over all the slots.

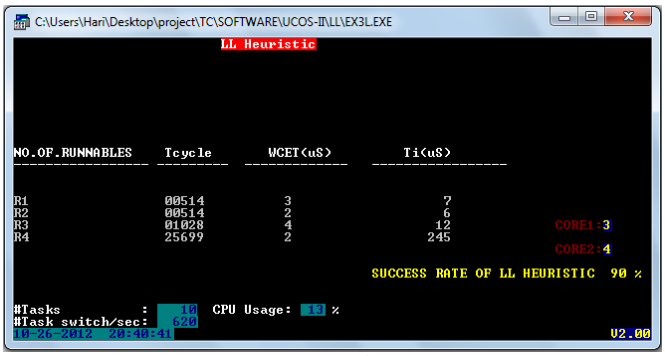

Fig. 3.Simulation result for LL heuristics.

Algorithm 3: LP heuristic.
Input: Runnable set *{Ri}*, *Ttic*, *Tcycle*

1) Sort runnables *Ri* such that *Ttic* ≤ *T*1 ≤ · · · ≤ *Tn* ≤ *Tcycle*.
2) *Twindow* = *Ttic*.
3) For *i* = 1. . . *n*
   a) *Twindow* = *LCM*(*Twindow*, *Ti*).
   b) In the first (*Ti/Ttic*) slots, look for the slot such that the highest load in the slots where *Ri* is periodically allocated in the (*Twindow/Ttic*) first slots is the lowest.
   c) Allocate *Ri* in every (*Ti/Ttic*) slot, starting from this slot.

Step 1 of Algorithm 3 runs in *O* (*n* log *n*). Step 3a runs in *O*(log *Tcycle*). Steps 3b and 3c, respectively, run in *O* (*n*(*Twindow/Ttic*)) ≤ *O*(*n*(*Tcycle/Ttic*)) and *O*(*n*(*Tcycle/Ti*)) ≤ *O*(*n*(*Tcycle/Ttic*)). As a result, the whole algorithm runs in *O* (*n* (log *n* + 2(*Tcycle/Ti*) + log *Tcycle*)).

Experiments show that these algorithms sometimes do not always perform well with runnable sets where a few runnables with low frequency have a very large WCET compared to other runnables. In practice, runnables with a large WCET tend to have a large period. As a result, runnables with a large WCET are usually processed late in the runnable allocation, which explains the load peaks. To reduce these peaks, the scheduling algorithm is improved by first processing some runnables with a large WCET. When the load is close to the harmonic schedulability bound, the algorithms remain efficient, in particular the LP, which successfully scheduled the 1000 random configurations of the test. This result suggests that the harmonic schedulability bound is also a good dimensioning criterion in the non-harmonic case.

T. Sivamani and R. Punitha, "Multicore ECU in automotive domain with FPP scheduling," *International Journal of Scientific and Technical Advancements*, Volume 1, Issue 4, pp. 45-49, 2015.

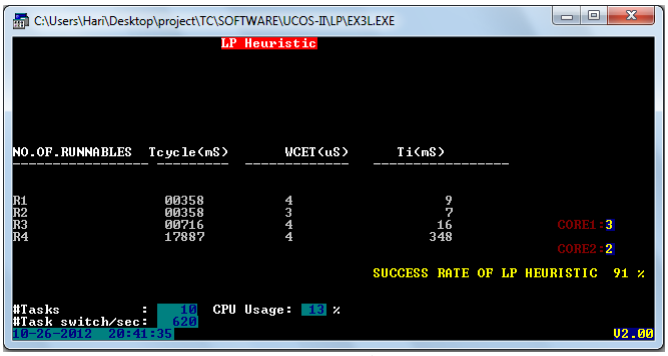

Fig. 4. Simulation result for LP heuristics.

Least crowded algorithm: In this algorithm, each child runnable is compared with its parent runnable. If the child runnable dominates a parent runnable, the child runnable is accepted as a parent runnable in the next generation. If a parent runnable dominates the child runnable, the child runnable is discarded and a new child runnable must be created. In the case that both are indifferent, the child runnable will be compared with an archive of so far best solutions. If it dominates any member in the archive, it will be a parent runnable in the next generation. But if the child runnable does not dominate any member in the archive, both parent runnable and child runnable are compared for their nearness (distance) to members of the archive. Task utilization rate= WCET/ Time period. If the child runnable resides in a least crowded region, it is accepted as a parent runnable and will be added to the archive. In the case that child runnable and parent runnable have the same nearness (distance) to the archive, one of them is selected depending upon the nanosecond priority manner. When the utilization rate decreases then the load on CPU also decreases to run.

Distribution of the load percentage over time:

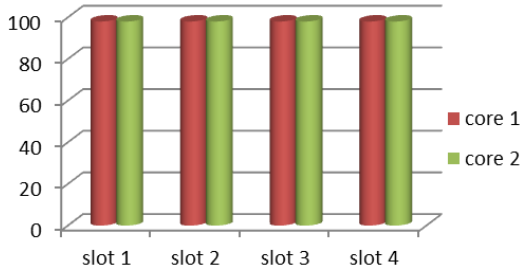

Fig. 5. Graphic shows the result of least crowded algorithm.

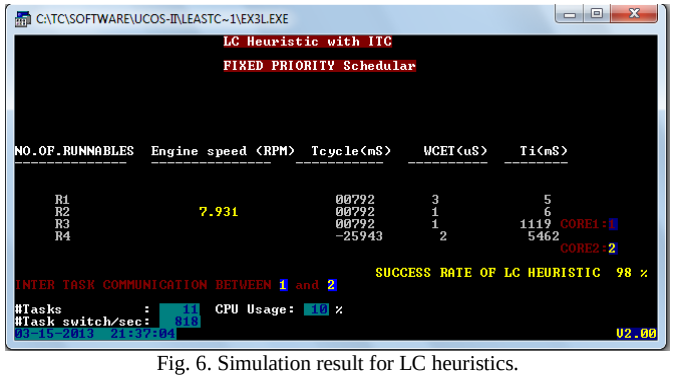

Fig. 6. Simulation result for LC heuristics.

IV. EXPERIMENTATIONS

Schedulability Performances and Robustness on Automotive ECUs

The goal is to assess the extent to which the schedulability bound, even if it has been derived in the harmonic case, can provide guidelines for the nonharmonic case. Precisely, we measure the success rate of the algorithms in the nonharmonic case at load levels such that feasibility would be ensured in the harmonic case. In the existing body gateway ECU, the set of task periods is close to be harmonic, because withdrawing only a few runnables ensures the harmonic property. To test the algorithms in a more difficult context, we build a "hard" nonharmonic case with more departure from the harmonic property.

Performance Scheduling Algorithm:

Success rate of Algorithm	WCET=500µs	CPU load in %
Success % of LL	90	13
Success % of LP	91	13
Success % of LP1σ	92	17
Success % of LC	98	10
Success % of LP1 σ with FPS	89	88

V. CONCLUSION

Multisource software and multicore ECUs will drastically change the electrical/electronic architectures and should enable more cost effective and more flexible automotive embedded systems. In our view, the OS protection mechanisms specified by AUTOSAR provide a sound basis for developing appropriate safety mechanisms and policies, despite the growing complexity and criticality of software functions. However, current design methodologies need to be adapted to this new context, and there is a wide range of technical problems to be solved. Among these issues are the design of the software architectures and the scheduling of the software components, which have been considered in this paper. The set of runnable sequencing algorithms proposed in this paper aims at uniformizing the load over time and thus increases the maximum workload schedulable on the CPU. The algorithms also provide guaranteed performance levels in some specific contexts. Experimentations on realistic case studies have confirmed that the algorithms are versatile and efficient in terms of CPU usage optimization.

REFERENCES

[1] Emadi, Y. Lee, and K. Rajashekara, "Power electronics and motor drives in electric, hybrid electric, and plug-in hybrid electric vehicles," *IEEE Transactions on Industrial Electronics*, vol. 55, no. 6, pp. 2237–2245, 2008.

[2] F. Mapelli, D. Tarsitano, and M. Mauri, "Plug-in hybrid electricvehicle: Modeling, prototype realization, and inverter loss reduction analysis," *IEEE Transactions on Industrial Electronics*, vol. 57, no. 2, pp. 598–607, 2010.

[3] D.-J. Kim, K.-H. Park, and Z. Bien, "Hierarchical longitudinal controller for rear-end collision avoidance," *IEEE Transactions on Industrial Electronics*, vol. 54, no. 2, pp. 805–817, 2007.

T. Sivamani and R. Punitha, "Multicore ECU in automotive domain with FPP scheduling," *International Journal of Scientific and Technical Advancements*, Volume 1, Issue 4, pp. 45-49, 2015.

ISSN: 2454-1532

[4] T. Bucher, C. Curio, J. Edelbrunner, C. Igel, D. Kastrup, I. Leefken, G. Lorenz, A. Steinhage, and W. von Seelen, "Image processing and behavior planning for intelligent vehicles," *IEEE Transactions on Industrial Electronics*, vol. 50, no. 1, pp. 62–75, 2003.

[5] N. Navet, A. Monot, B. Bavoux, and F. Simonot-Lion, "Multisource and multicore automotive ECUs—OS protection mechanisms and scheduling," *in Proceedings IEEE International Symposium on Industrial Electronics (ISIE)*, pp. 3734–3741, 2010.

[6] A. Burchard, J. Liebeherr, Y. Oh, and S. H. Son, "New strategies for assigning real-time tasks to multiprocessor systems," *IEEE Transactions on Computers*, vol. 44, no. 12, pp. 1429–1442, 1995.

[7] Y. Oh and S. Son, "Fixed-priority scheduling of periodic tasks on multiprocessor systems," Department of Computer Science, University of Virginia, Charlottesville, VA, Tech. Rep. CS-95-16, 1995.

[8] S. Lauzac, R. Melhem, and D. Mossé, "An improved rate-monotonic admission control and its applications," *IEEE Transactions on Computers*, vol. 52, no. 3, pp. 337– 350, 2003.

T. Sivamani and R. Punitha, "Multicore ECU in automotive domain with FPP scheduling," *International Journal of Scientific and Technical Advancements*, Volume 1, Issue 4, pp. 45-49, 2015.