Energy-aware Resource Allocation in Multi-mode Automotive Applications with Hard Real-Time Constraints

Piotr Dziurzanski, Amit Kumar Singh and Leandro Soares Indrusiak
Department of Computer Science, University of York, Deramore Lane, Heslington, York, YO10 5GH, UK.
{Piotr.Dziurzanski, Amit.Singh, Leandro.Indrusiak}@york.ac.uk

***Abstract*—This paper presents an energy aware resource allocation approach that benefits from modal nature of hard-real time systems under consideration. The modal nature of the considered applications made it possible to decrease the number of active cores consuming high power in certain modes or to switch into core states with lower power consumption, which lead to considerable energy savings while still not violating any of the timing constraints. For the considered automotive use case, the number of required cores has been decreased by up to 75% in a particular mode and relatively low amount of data is to be migrated during the mode change. The trade-off between the amount of data to be migrated and energy dissipation in the subsequent state is also analysed.**

I. Introduction

In contemporary cars the number of electronic control units (ECUs) sometimes reaches even 100. Since building an embedded system comprised of such number of devices is rather challenging, the automotive industry gradually resigns from their paradigm of using a separate unit for each functionality [10]. This has led to the requirement of placing a number of ever more sophisticated functionalities in one chip, which has resulted in appearance of multi-core ECUs [19].

Based on the AUTOSAR (AUTomotive Open System ARchitecture) standard [1], atomic software components, named *runnables*, are mapping statically (i.e. determined during design time) into cores since it is less complex and more predictable than dynamic resource allocation [8]. The runnables in automotive systems are usually confined with hard real-time constraints. Consequently, the cores have to execute all the tasks on time even for their worst-case execution behavior, where they take the worst-case execution time (WCET), which is usually much higher than the average execution time [18]. The difference between the worst and average task execution times can be decreased by exploiting modal nature of such applications. The modal nature determines the number of ways in which the applications can behave, referred to as *modes*, and it can be known at design time.

Some modes of an example gasoline engine, together with its relation with its throttle, RPMs and acceleration pedal are illustrated in Fig. 1 (idea of this picture has been taken from [11]). The starting mode, PowerUp, describes the situation when the the key has been just inserted into the ignition. In the following Stalled mode the throttle is still closed. In cranking mode the engine starts, so the number of RPM increses up to the idle RPM level. Then the engine stays in the Idle mode as long as the driver does not push the accelerator pedal. Thereafter, the current mode changes to Drive, where the throttle opens and the number of RPM grows above the idle RPM level. If each mode is analysed independently, the average execution time may be closer to the WCET determined for that mode [13].

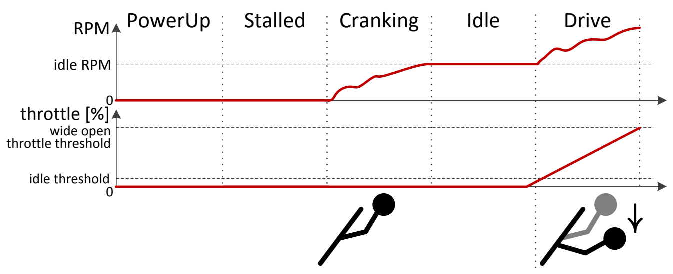

Fig. 1. Modes in DemoCar

In different modes, the contexts of runnables that are executed on different cores need to be migrated from one core to another. This sets additional requirements for the available communication bandwidth. The process of mode switching (e.g. from PowerUp to Drive) usually incurs overhead (both in execution time and energy), if allocation needs to be changed. This overhead needs to be taken into account at run-time to decide whether to change the allocation for the next mode or not.

In hard real-time systems, even during the mode switching process, it is essential to satisfy all the timing constraints, i.e. the migration time of tasks must be time bounded [6]. Therefore, the worst case switching time has to be assumed to provide the timing guarantees.

To enable energy awareness in automotive systems, Dynamic Voltage and Frequency Scaling (DVFS) technique omnipresent in CMOS circuits can be exploited [12]. It benefits from the fact that their dynamic (or switching) power P is proportional to the square of core supply voltage V and its clock frequency f, $P \propto fV^2$. Contemporary circuits usually follow the Advanced Configuration and Power Interface (ACPI) open standard, defining processor states known as P-states. In the highest P-state, P0, a processor works with the highest voltage and frequency level, but offers the best performance. In P1 and other modes, the processor works slower, but dissipates less energy. Since any reduction of core voltage requires an adequate decrease of the clock frequency, some trade-off between energy savings and computation performance is expected. Some guidance in real-time systems stems from the fact that there is usually no additional benefits from faster task execution as long as it is before the deadline. Therefore, slower executions at lower voltage and frequency levels can be performed while meeting the deadline in order to save energy consumption.

Contribution: In this paper, we consider various modes of automotive applications and determine the best allocation for each mode by employing a genetic algorithm (GA) based approach. The approach performs optimization for energy dissipation and migration cost in terms of the context length of the transmitted runnables. To guarantee that the mode switching migration finishes in the required time, the traditional schedulability analysis is used to determine the necessary network bandwidth.

This paper is organised as follows. In the next section, the state-of-the-art solutions are reviewed. Then, in Section III, the applied application and platform models together with the problem formulation are described. In Section IV the steps of the proposed design flow are presented. They are experimentally evaluated in Section V using an engine ECU code named *DemoCar* from Robert Bosch GmbH. The paper is concluded in Section VI.

II. Related works

Exploiting the knowledge about distinguishable operating modes in a system is tempting and thus modal systems are an increasingly popular subject in research. Since the number of possible scenarios is typically prohibitively high [16], a number of research activities aims at developing design-time (off-line) heuristics to reduce the number of operating points. This Design Space Exploration (DSE) process can be carried out using classic heuristic techniques for clustering modes so that their final number is manageable. Then during run-time of that system, a run-time manager (RTM) determines the current mode out of an explicitly given set by observing some variables of the model [11].

Two different mapping approaches are proposed in [14], but they do not allow task migration, i.e. once a task is assigned to a processing core, it remains there until its computation is finished. In contrast, Benini et al. [2] allowed tasks to migrate between processing cores when the envisaged performance gain is higher than the precomputed migration cost.

The possible modes and transitions between them can be shown in a formal way in order to analyse the worst case switching time between two modes. An example formal way could be to use Finite State Machines (FSMs), as proposed in [13]. This facilitates to identify all the allowed modes and the transitions between them, and to check the cost of mode switchings. In [5], for H.264 decoder, an average switching time overhead between two modes has been measured to be equal to 0.2% of the total system time. This slight value has been caused due to a low number of existing modes, obtained due to the clustering, and thus relatively lower switching. In [17], the authors suggest to map as many tasks as possible to the same core in various modes to avoid the data or code items to be moved between different resources when switching between modes. However, this condition does not take into consideration different context sizes of the tasks. In the proposed approach, we minimize the amount of data to be migrated instead.

To guarantee hard real-time during task migration, a methodology is proposed in [9]. However, a costly schedulability analysis is performed during run-time. Further, experiments supporting their proposed approach are not provided, but one may predict that the overload of that dynamics could be considerable.

The approach closest to the approach described in this paper is that of [11], where mode transition points in an engine management system are identified and it is shown that a load distribution by mode-dependent task allocation is better balanced in comparison with a static task allocation. However, in contrast to our approach, the task migration costs have not been considered.

In our prior work [4], an earlier version of the proposed approach has been presented. In that version, DVFS has not been exploited, thus only a single objective genetic algorithm has been employed to find a quasi-optimum mapping, whereas in this paper we use a two-objective genetic algorithm and also encode core voltage/frequency levels into inviduals. The contribution of that paper has focused mainly on the issue of schedulability in each mode and also during mode changes, whereas in this paper we present multiple solutions in a form of a Pareto frontier to choose a solution representing a trade-off between migration time and the energy consumed in the future mode. Also, different modes in the DemoCar example have been indentified.

The close observation of literature survey indicates that designing real-time systems with distinguishable operating modes has been mainly limited to soft timing constraints, which means deadline violations could occur. To the best of our knowledge, there is no proposal of any other method guaranteeing no hard deadline violation during task migrations required to move from one mode to another while applying low cost schedulability analysis to check the feasibility of the task migration process.

III. System model

A. Application model

In this work we assume application model is consistent with the AUTOSAR standard [1]. A taskset Γ is comprised of an

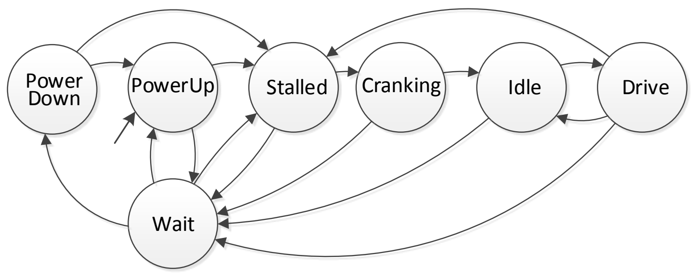

Fig. 2. FSM describing modes in DemoCar

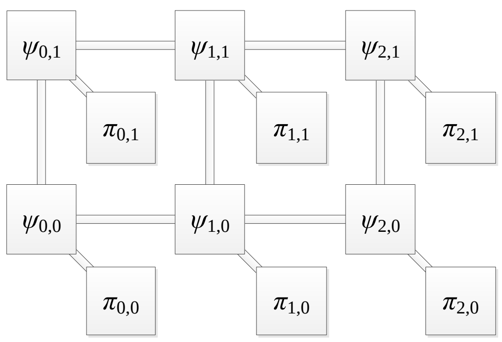

Fig. 3. An example many-core system platform

arbitrary number of periodic runnables, $\Gamma = \{\tau_1, \tau_2, \tau_3, \ldots\}$, grouped in tasks with hard real-time constraints. Their properties depend on the current mode μ of the application. The j-th occurrence (j-th job) of runnable τ_i is denoted with $\tau_{i,j}$. The taskset is known in advance, including the WCET of each runnable, $C_{i,\mu}$ in every mode μ, its period T_i, priority P_i and its relative deadline D_i equal to this period. Runnables are atomic schedulable units communicating each other with so called *labels*, which are memory locations of a particular length. The order of read and write operations to labels denotes the runnable dependencies, as the write operation to a particular label should be completed before its reading. We assume that the labels are stored in the same node that the runnable that reads these labels. If more than one runnable mapped to different cores read from the same label, its content is to be replicated to all the reading nodes and the writer should update the label value at all the locations. It means that the writer is aware of all its readers and knows their locations in all the possible modes.

All possible modes of the application together with the allowed transitions between them are known. They may be described using an FSM, similar to the one presented in Fig. 2, where 7 modes and 16 possible transitions are shown. Deadlines for mode changing time between each neighbouring pair of modes shall be also provided.

B. Platform model

The hardware platform assumed in this paper is a mesh Network on Chip (NoC) with a certain number of cores $\pi \in \Pi$ and routers $\psi \in \Psi$, as shown in example in Fig. 3. Each link is modelled as a single resource, so, for example, to transfer a portion of data from $\pi_{0,1}$ to appropriate sink $\pi_{2,0}$ we need such resources allocated simultaneously: $\pi_{0,1} - \psi_{0,1}$, $\psi_{0,1} - \psi_{1,1}$, $\psi_{1,1} - \psi_{2,1}$, $\psi_{2,1} - \psi_{2,0}$, $\psi_{2,0} - \pi_{2,0}$.

In every mode, each runnable is mapped to one core and a label is stored in the local memories of the cores requesting that label. Data transfer overhead is taken into consideration, assuming constant time for transferring a single flit (Flow control digIT, a piece of a network package whose length usually equals the data width of a single link) between two neighbouring cores if no contentions are present. Timing constants for packet latencies while traversing one router and one link are denoted as d_R and d_L, respectively. The priority of data transfer packets are assumed to be equal to the priority of the runnable sending them.

The processing cores can operate under a given set of voltage and frequency levels, but the links have no P-states.

C. Problem formulation

Given a platform and an application model with a defined set of operating modes, the problem is to determine schedulable mappings for each mode so that the amount of data to be migrated during allowed mode changes and energy consumed by the platform are minimized. Since these two criteria may be contradictory, a trade-off between them shall be illustrated with a Pareto frontier.

During mode changing, the taskset should be still schedulable despite the additional network traffic generated by the task migrations. The neighbouring modes with similar runnables' execution time can be clustered to decrease the frequency of task migrations. Deadlines for mode changing time between each neighbouring pair of modes must not be violated.

IV. Proposed approach

In this section, steps of the proposed design flow are described. Since it has been assumed that the tasksets of the considered application are known in advance, it is possible to perform the majority of the required computations statically. Consequently, the mapping problem can be split into two stages: off-line (static) and on-line (dynamic), as shown in Fig. 4. The computation time of the off-line part is not crucial and thus heuristics with even high complexity, such as genetic algorithms, may be used for runnable and label mappings.

During the application run-time, detection of the current mode is assumed to be done by observing a certain variable. When a value of this variable has been changed, the current runnable and label mapping might need to be changed. The mappings have been identified at the design time while trying to minimize the amount of data to be migrated during the static mapping and P-state selection step. Schedulability analysis

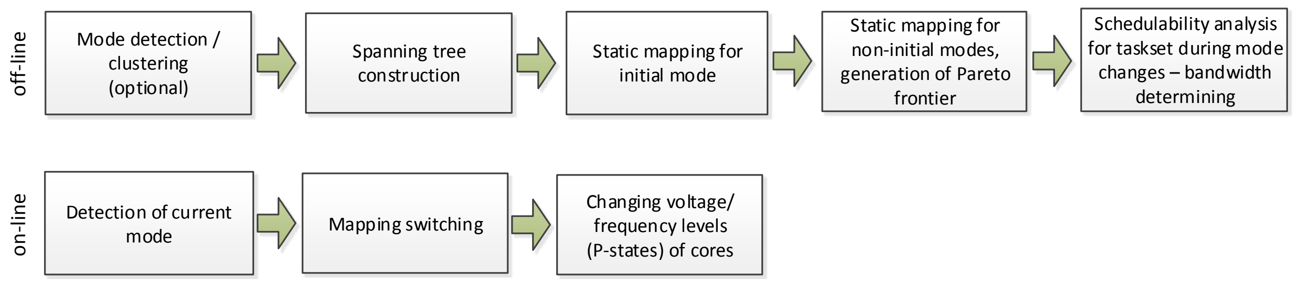
off-line
Mode detection / clustering (optional)
Spanning tree construction
Static mapping for initial mode
Static mapping for non-initial modes, generation of Pareto frontier
Schedulability analysis for taskset during mode changes – bandwidth determining
on-line
Detection of current mode
Mapping switching
Changing voltage/ frequency levels (P-states) of cores

Fig. 4. Steps of the proposed energy aware dynamic resource allocation method benefiting from modal nature of applications

guarantees that even the worst case switching time does not violate the deadline required for mode changes. If such violation is unavoidable, either the states can be clustered, or the network bandwidth is to be increased.

A. Static mapping

1) Initial mode: Algorithm 1 presents a pseudo-code of a genetic algorithm that can be used to identify a mapping for the initial identified mode. The algorithm ensures that no deadline violation occurs under the chosen allocation. We propose to use two fitness functions - measuring (i) the number of deadline violations and (ii) the total energy dissipated by the resources. The first fitness function value is of primary importance, as in a hard real-time system no deadline violation is allowed. However, among fully schedulable mappings, the one leading to a lower dissipated energy is chosen.

Each chromosome in the genetic algorithm contains genes of two types, as shown on the top of Fig. 5. The first n genes indicate the target cores for n runnables and the remaining $|\Psi|$ genes (for a mesh NoC $|\Psi| = x \cdot y$, where x and y are the mesh dimensions) specifies the P-states of the consecutive cores.

In the algorithm, the following two main steps can be singled out.

Step 1. Initial population initialisation (line 1). An arbitrary number of random task mappings (individuals and P-states) is created.

Step 2. Creating a new population (lines 3-10). For each individual, values of the two fitness functions (the number of deadline violations and dissipated energy (lines 3-4)) are computed. Individuals with the same number of deadline misses are grouped together (line 5). The groups are then sorted with respect to the number of deadline violations in the ascending order (line 6). Inside each group, individuals are sorted according to their growing dissipated energy (line 7). The tournament selection is then performed, where individuals from a group with lower number of deadline violations are always preferred, whereas among individuals from one group the one with the lowest dissipated energy is to be chosen (line 8). The individuals winning the tournament are then combined

Algorithm 1: Pseudo-code of no deadline violation with energy minimisation algorithm for the initial mode mapping

```
inputs  : Workload $\Gamma$;
          Resource set $\Pi$;
outputs : Task mapping; Core P-states;
1  Choose an initial random population of task
     mappings and P-states
2  while not termination condition do
3      Evaluate the number of deadline violations;
         //criterion (i)
4      Evaluate the dissipated energy; //criterion (ii)
5      Create clusters of individuals with the same
         number of deadline violations;
6      Sort the clusters by increasing number of
         deadline violations;
7      Sort individuals in each cluster w.r.t the dissipated
         energy ;
8      Perform tournament selection; //criterion (i) has
         higher priority than criterion (ii)
9      Generate individuals using crossover and
         mutation;
10     Create a new population with the best found
         mappings;
   end
```

using a typical crossover operation and mutated (line 9). Then, a new population is created from these individuals (line 10). Step 2 is repeated in a loop as long as a termination condition is not fulfilled, which can be a maximal number of generated populations or lack of improvement in a number of subsequent generations.

2) Non-initial modes: As mentioned earlier, it is of primary importance to migrate as little data as possible during mode changes to minimise the migration time and energy. However, it may be beneficial to migrate more data if the energy consumed in the next mode is much lower than the migration energy. Thus there could be some trade-off between migration

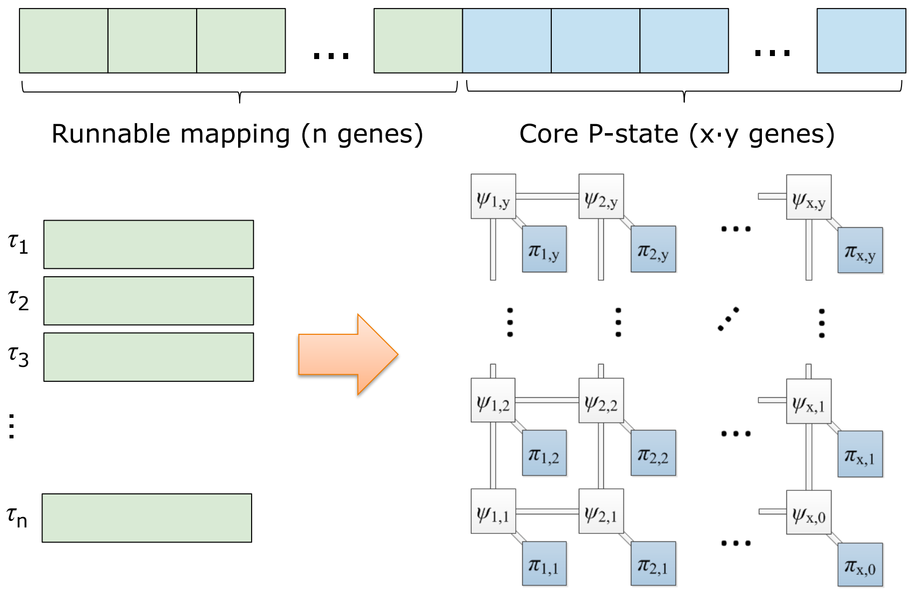

Fig. 5. Genes in chromosomes

data (or time) and energy consumption in the next mode. It is role of a designer to choose a proper solution from the Pareto frontier.

Each application A includes a set of tasks and can be represented with a vector comprised of p runnables $A = [\tau_1, \ldots, \tau_p]$. Platform Π is composed of s processing cores, $\Pi = \{\pi_1, \ldots, \pi_s\}$. A mapping M is a vector of p core locations, $\mathrm{M} = [\pi_{\tau_1}, \ldots, \pi_{\tau_p}]$, where each element corresponds with the appropriate element of A and can be substituted with any element of set Π.

To perform optimization for migration cost that considers the context length of the transmitted runnables, weight vector W is introduced. Each element of this vector $\mathrm{W} = [w_{\tau_1}, \ldots, w_{\tau_p}]$ is equal to the amount of data that has to be transferred when a particular runnable is migrated, including the labels to be read or written.

Let M_α and M_β be sets of mappings (i.e. set of Ms) that are fully schedulable in a given system in mode α and β, respectively. The elements of the difference vector $D_{m_\alpha, m_\beta} = [d_{\tau_1}, \ldots, d_{\tau_p}]$ indicate which runnables are to be migrated when the mode is changed from α to β. Each element d_δ, $\delta \in \{\tau_1, \ldots, \tau_p\}$, takes value 1 if the particular runnable/label is allocated to different cores in mappings $m_\alpha \in M_\alpha$ and $m_\beta \in M_\beta$, and 0 otherwise:

$$d_\delta = \begin{cases} 1, & \text{if } m_{\alpha,\delta} \neq m_{\beta,\delta}, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

where $m_{\alpha,\delta}$ and $m_{\beta,\delta}$ denote the δ-th element of vectors m_α and m_β, respectively. The migration cost c between two modes α and β is then computed in the following way

$$c_{m_\alpha, m_\beta} = \mathrm{D}_{m_\alpha, m_\beta} \cdot \mathrm{W}^{\mathrm{T}}. \tag{2}$$

Algorithm 2: Pseudo-code of a migration data transfer and energy minimisation algorithm

```
inputs  : A spanning tree ST based on Finite State
          Machine (FSM) describing the system modes
          with transaction probabilities;
          W - size of each runnable memory footprint;

outputs : Runnable and label mapping for each mode;
          P-states for cores in each mode;

1   Select the initial state of ST and assign it to $\alpha$;
2   Find a set of schedulable mappings $\mathrm{M}_\alpha$;
3   Select $m_\alpha \in \mathrm{M}_\alpha$ that consumes the lowest
     amount of energy;
4   forall $\beta$ being a direct successor of $\alpha$ in ST do
5   |   FindMappingMin($\alpha$, $\beta$, $m_\alpha$);
    end

     FindMappingMin($\alpha$, $\beta$, $m_\alpha$)
1.1  Find a Pareto frontier of schedulable mappings
      $\mathrm{M}_\beta$ minimizing criterion Equation (2) and
      energy consumption in $\beta$ using W
1.2  Select $m_\beta \in M_\beta$ wrt design priorities
1.3  forall $q$ being a direct successor of $\beta$ in ST do
1.4  |   FindMappingMin($\beta$, $q$, $m_\beta$)
    end
```

A recursive greedy algorithm for reducing the amount of data transferred during mode changes is presented in Algorithm 2.

Since some cycles are likely to occur in a graph representing

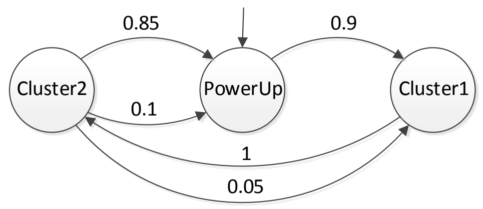

Fig. 6. FSM describing clustered modes in DemoCar

the Finite State Machine describing transitions between modes, a spanning tree (ST) is to be built, whose branches are labelled with the transitions probabilities (like in Fig. 6). Then the mode corresponding to the initial state of the FSM is selected as the current mode (line 1). For this mode, a set of schedulable mappings is generated, e.g. with Algorithm 1 (line 2). If more than one schedulable mapping is found, the one leading to the lowest energy consumption is selected (line 3). Then for each direct successor of the ST node corresponding to FSM initial state, the *FindMappingMin* procedure is executed (lines 4 and 5).

In the *FindMappingMin* procedure, a Pareto frontier of schedulable mappings for that successor node is found using two criteria: i) minimal migration cost criterion Equation (2) and ii) minimal energy dissipated in the next mode (line 1.1). The most suitable schedulable mapping is chosen from the Pareto frontier based on the design priorities (line 1.2). The *FindMappingMin* procedure is then recursively run for each direct successor of the ST node provided as the function parameter (lines 1.3 and 1.4).

More mappings could be delivered to the *FindMappingMin* procedure to browse a larger search space by skipping lines 3 and 1.2 in the algorithm and providing all elements of M_α instead of just one. It is the role of a designer to set priorities between the migration time and energy dissipation to select the most suitable solution from the Pareto frontier.

B. Schedulability analysis

The proposed task mapping technique aims to benefit from modal nature of applications, but it also possess new challenges. If the modes are treated independently from each other, the end-to-end schedulability of runnables and packet transmission in each mode can be analysed using equations from [15].

It is the instant of transition between the modes that requires special attention. During transition, the task migration time can be computed with equations from [15], where the packet size is equal to the sum of the header length and the size of the payload including the whole context of runnables and labels to be migrated. To guarantee taskset schedulability during migration, we propose to treat a migration process as any other asynchronous process in schedulability analysis, i.e. to use so-called *periodic servers*, which are periodic tasks executing aperiodic jobs. When a periodic server is executed, it processes pending task migration. If there is no pending migration, the server simply holds its capacity. Similarly to [9], we split a runnable context into two parts: i) invariant, which is not modified at runtime, and ii) dynamic, including all volatile memory locations. We assume that an upper bound of the dynamic part size of all runnables is known in advance. This part shall be migrated at once using the last instance of the periodic server. It means that the local memory locations that can be modified by the runnable must not be precopied, but migrated after the last execution of the runnable in the old location. This requirement can influence the minimum periodic server size (i.e. the time allocated to it by a scheduler in each period) and, consequently, the network bandwidth, as it must be then wide enough to guarantee migration of dynamic part before the next runnable execution (in the new location).

In the proposed approach, any kind of periodic servers can be used, however, the trade-off between implementation complexity and ability to guarantee the deadlines of hard real-time tasks, as described for example in [3], shall be considered.

C. On-line usage

In the proposed approach, only two steps are performed on-line: *Detection of current mode* and *Mapping switching*.

We assume that the system modes are defined explicitly and there is a possibility of determining the current mode by observing some system model variables[1], similarly to [11].

When the mode change is requested, an agent residing in each core prepares a set of packages with runnables to be migrated via the network. This agent is configured statically and is equipped with a table with information about runnables that need to be migrated during a particular mode change. Then the precopy of these runnables is performed. In the following hyperperiods, runnables are transported using periodic servers of the length determined statically using schedulability analysis, as described earlier. The agent is aware of the number of periodic server instances that have to be used during the whole migration process, and have the volatile portion of the context identified. If this instance number elapses, the runnables that have been migrated are killed on the earlier core.

Simultaneously, the same agent can receive migration data from other agents in the network. After the appropriate number of hyperperiods, the contexts of these runnables are fully migrated and are ready to be executed by the operating system.

The details of the agent depend on the underlying operating system. Regardless its implementation, *Detection of current mode* shall be characterised with low computational complexity and thus shall impose low overhead for the system during run-time. The number of the hyperperiods required for performing task migration during *Mapping switching* depends on the size of runnables and labels to be transferred, mappings, and network bandwidth, in particular flit size and timing

[1]In DemoCar such variable is named *_sm* and is stored in runnable *OperatingModeSWCRunnableEntity*.

constants for packet latencies while traversing one router and one link d_R and d_L.

V. Experimental results

As an example application, we consider a lightweight engine control system named DemoCar. It consists of 18 runnables and 61 labels. All runnables are periodic and combinational, i.e. their outputs depend only on input values. In Fig. 2, 7 identified modes of this application are presented. These modes have been identified by inspecting the code of the runnable named *OperatingModeSWC*, which computes values of transaction and output functions of the FSM steering this engine.

The transitions between modes: *Stalled*, *Cranking*, *Idle*, *Drive*, are to be performed between two consecutive executions of their runnable occurrences, which is upperbounded with 5ms for 9 runnables. Since performing task migration during such short time window would require a bandwidth of considerable size, these modes have been clustered into *Cluster1* (Fig. 6). For a similar reason, *Wait* has been clustered with *PowerDown* into *Cluster2*. Finally, three modes can be identified after the clustering step: *PowerUp*, *Cluster1* and *Cluster2*, as presented in Fig. 6.

The energy consumption of the multi-core system considered in this paper has been determined using the technique described in [7]. They are averages obtained during a series of simulations.

The processing core consumes 2.08E-4μJ when idle and 3.74E-4μJ when busy. An idle link dissipates 17.86E-6μJ whereas a link transporting a package dissipates 46.10E-6μJ. The core energy has been scaled using relation $P \propto fV^2$ and P-states, where the maximum voltage/frequency level has been assigned.

For the *PowerUp* (initial) mode of DemoCar to be executed on a multi-core embedded system, we estimate makespan and number of violated deadlines during one hyperperiod (i.e. the least common multiple of all runnables' periods) by allocating runnables and labels to different cores.

The size of the NoC mesh has been initially configured as 2x2 with no idle cores, since this size had been earlier checked (also using Algorithm 1) and is large enough to execute DemoCar in the most computational intensive mode, *Cluster1*, not violating any of its timing constraints. The flit size has been fixed to 16 bits. The genetic algorithm is executed again to perform assignment of tasks to cores with timing characteristics for the initial *PowerUp* mode. The genetic algorithm has been configured to generate 100 generations of 20 individuals each. The first fully schedulable allocation has been found in the 1st generation, which suggests that it might be possible to allocate the taskset to a lower number of cores.

After performing further search it has appeared that the taskset in the initial mode is schedulable even when mapped to one (out of four) active core. The lowest makespan for the NoC with three idle cores is equal to 8622μs. The energy consumed in this mode equals to 3093.01μJ per hyperperiod

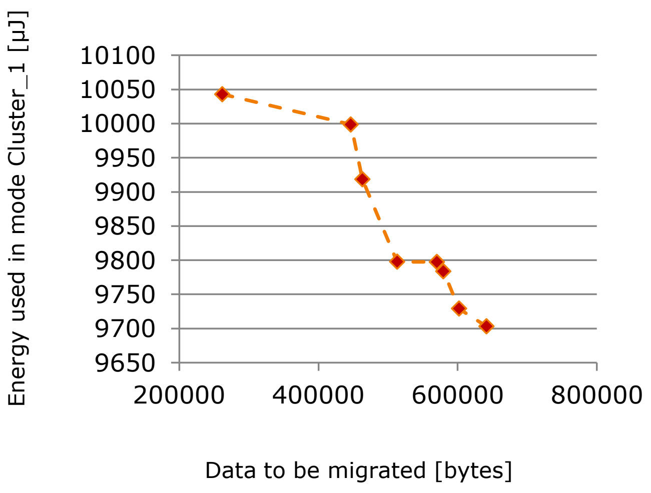

Fig. 7. Pareto curve illustrating the trade-off between minimal amount of data to be migrated and minimal energy consumed in the next mode

(100 ms). Thus, thanks to the modal approach, one can switch off 75% of the cores while the application is in the *Cluster1* mode.

Regardless of the mode, the application has been mapped in a 2x2 mesh Network on Chip without deadline violations. For the *PowerUp* mode, schedulable mappings have been found even if three of the four NoC cores remain idle. It means that in this mode three cores can be switched off, leading to considerable energy savings. Similarly, two cores can remain idle in the *Cluster2* mode. However, despite intensive search using a genetic algorithm, all four cores are needed in the *Cluster1* mode to have the taskset fully schedulable. Thus, when the current mode changes from *PowerUp* to *Cluster1*, three cores have to be activated, whereas two cores can be switched off after leaving the *Cluster1* mode.

Next we focused on the transition between the *PowerUp* and *Cluster1* modes. For *PowerUp*, only one core is active and thus all runnables are to be mapped to the only active core. However, in other cases a larger set of mappings that are fully schedulable on active NoC cores would have been identified. A Pareto frontier using two criteria: minimal amount of data to be migrated and minimal energy consumed in the next mode has been constructed and drawn in Fig. 7. If energy dissipation is crucial for the design and longer switching time can be accepted, the rightmost solution from the Pareto curve shall be chosen. On the contrary, the leftmost solution from the Pareto curve is appropriate for the system with switching time more bounded, where some energy loss may be tolerated. The remaining 6 solutions form a compromise between these two extremes.

Assuming that minimal energy consumption is crucial for the system, the solution leading to consumption of 9719.45μJ (in the next mode) should be chosen. Then, using the same priority, the mapping in the *Cluster2* mode would consume

TABLE I
NUMBER OF HYPERPERIODS (100MS) REQUIRED FOR SWITCHING BETWEEN MODES *PowerUp* TO *Cluster1* IN DEMOCAR DEPENDING ON ROUTER (d_R) AND ONE LINK LATENCIES (d_L)

		No. of hyperperiods	
d_R [ns]	d_L [ns]	solution A	solution B
100	200	1	1
100	400	1	2
200	500	1	2
400	800	2	3
500	1000	2	3

5909.37μJ.

We also have evaluated the number of hyperperiods required to migrate tasks from *PowerUp* to *Cluster1*, depending on constants d_R and d_L are presented in Table I. Two extreme solutions from the Pareto frontier illustrated in Fig. 7 are analysed: A is the mapping with the lowest amount of data to be migrated, B is the solution with the lowest energy dissipated in mode *Cluster1*. The hyperperiod length for DemoCar equals 100ms and this time is enough to migrate all data when the router and link latencies are equal to 50 and 100ns, respectively.

VI. CONCLUSIONS

An approach for task migration in a multi-core network-based embedded system has been proposed as a way to decrease the number of cores needed for guaranteing safe execution of a hard real-time software. Applying different value/frequency levels (P-states) to cores facilitates decreasing of energy dissipation even further. The poposed approach is comprised of steps to be performed statically (off-line) and during runtime (on-line). The approach has been illustrated with DemoCar, a simple gasoline ECU. A Finite State Machine describing mode changes has been extracted from its code and transaction probabilities have been identified during simulation. The closely related modes have been merged into clusters. A genetic algorithm has been used to determine the runnable-to-core mapping for the initial mode. Similarly, a multi-objective genetic algorithm minimizing the migrated data and the energy dissipated in the next state has been used. Each Pareto-optimal solution determines the runnables to be migrated when a change of the current mode is requested. The migration time has been evaluated using schedulability analysis depending on the network bandwidth.

The proposed approach requires development of an agent realising the migration process. Since its architecture details depend on the underlying operating system, its implementation and evaluation in real embedded environments are planned as a future work.

ACKNOWLEDGEMENT

The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 611411.

The authors would like to thank Björn Saballus from Robert Bosch GmbH for providing the DemoCar use case.

REFERENCES

[1] AUTOSAR: AUTomotive Open System ARchitecture, http://www.autosar.org, 2015.

[2] L. Benini, D. Bertozzi, and M. Milano, "Resource management policy handling multiple use-cases in MPSoC platforms using constraint programming," Logic Programming, Springer Berlin Heidelberg, pp. 470–484, 2008.

[3] R.I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," ACM Comput. Surv., vol. 43, no. 4, art. 35, pp. 1–44 , 2011.

[4] P. Dziurzanski, A.K. Singh, L.S. Indrusiak, and B. Saballus, "Hard real-time guarantee of automotive applications during mode changes," Proceedings of the 23rd International Conference on Real Time and Networks Systems (RTNS 2015), pp. 161–170, 2015.

[5] S.V. Gheorghita et al., "System-scenario-based design of dynamic embedded systems," ACM Trans. Des. Autom. Electron. Syst., vol. 14, no. 1, art. 3, pp. 1–45, January 2009.

[6] J.R. van Kampenhout, "Deterministic Task Transfer in Network-on-Chip Based Multi-Core Processors," Computer Engineering, no. 18, 2011.

[7] K. Latif et al., "An Integrated Framework for Model-Based Design and Analysis of Automotive Multi-Core System," Forum on specification & Design Languages, FDL'15, Work-in-Progress Session, Barcelona - Spain, 2015.

[8] A. Monot, N. Navet, B. Bavoux, and F. Simonot-Lion, "Multisource Software on Multicore Automotive ECUs - Combining Runnable Sequencing with Task Scheduling," IEEE Transactions on Industrial Electronics, vol. 59, no. 10, pp. 3934–3942, 2012.

[9] P. Munk, B. Saballus, J. Richling, and H.U. Heiss, "Position Paper: Real-Time Task Migration on Many-Core Processors," 28th International Conference on Architecture of Computing Systems (ARCS'15), pp. 1–4, 2015.

[10] M. Di Natale and A.L. Sangiovanni-Vincentelli, "Moving From Federated to Integrated Architectures in Automotive: The Role of Standards, Methods and Tools," Proceedings of the IEEE, vol. 98, no. 4, pp. 603–620, 2010.

[11] J. Park et al., "Mode-Dynamic Task Allocation and Scheduling for an Engine Management Real-Time System Using a Multicore Microcontroller," SAE Int. J. Passeng. Cars - Electron. Electr. Syst., vol. 7, no. 1, pp. 133–140, 2014.

[12] E. Quinones, J. Abella, F. J. Cazorla, and Mateo Valero, "Exploiting intra-task slack time of load operations for DVFS in hard real-time multi-core systems," SIGBED Rev. vol. 8, no. 3, pp. 32–35, 2011.

[13] L. Schor, I. Bacivarov, D. Rai, H. Yang, S.H. Kang, and L. Thiele, "Scenario-based design flow for mapping streaming applications onto on-chip many-core systems," ACM International conference on Compilers, architectures and synthesis for embedded systems, pp.71–80, 2012.

[14] A. Schranzhofer, J.J. Chen, and L. Thiele, "Dynamic Power-Aware Mapping of Applications onto Heterogeneous MPSoC Platforms," IEEE Trans. on Industrial Informatics, vol. 6, no. 4, pp. 692–707, November 2010.

[15] Z. Shi and A. Burns, "Real-time communication analysis for on-chip networks with wormhole switching," ACM/IEEE International Symposium on Networks-on-Chip (NOCS'08), pp. 161–170, 2008.

[16] A.K Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: survey of current and emerging trends," Proceedings of the 50th Annual Design Automation Conference (DAC), 2013.

[17] S. Stuijk, M. Geilen, B. Theelen, and T. Basten, "Scenario-aware dataflow: Modeling, analysis and implementation of dynamic applications," International Conference on Embedded Computer Systems (SAMOS'11), pp. 404–411, 2011.

[18] R. Wilhelm et al., "The worst-case execution-time problem-overview of methods and survey of tools," ACM Trans. Embed. Comput. Syst., vol. 7, no. 3, art. 36, pp. 1–53, 2008.

[19] D. Zhu and C. Qian, "Challenges in Future Automobile Control Systems with Multicore Processors," Workshop on Developing Dependable and Secure Automotive Cyber-Physical Systems from Components, 2011.