RESEARCH

Multi-criteria Resource Allocation in Modal Hard Real-Time Systems

Piotr Dziurzanski*, Amit Kumar Singh and Leandro Soares Indrusiak

*Correspondence: piotr.dziurzanski@york.ac.uk
Department of Computer Science, University of York, Deramore Lane, Heslington, YO10 5GH York, UK
Full list of author information is available at the end of the article

Abstract

In this paper, a novel resource allocation approach dedicated to hard real-time systems with distinctive operational modes is proposed. The aim of this approach is to reduce the energy dissipation of the computing cores by either powering them off or switching them into energy-saving states while still guaranteeing to meet all timing constraints. The approach is illustrated with two industrial applications, an engine control management and an engine control unit. Moreover, the amount of data to be migrated during the mode change is minimised. Since the number of processing cores and their energy dissipation are often negatively correlated with the amount of data to be migrated during the mode change, there is some trade-off between these values, which is also analysed in this paper.

Keywords: hard real-time systems; modal systems; task allocation; task migration; low power design; dynamic voltage and frequency scaling

Introduction

Electronic control units (ECUs) have become key components of contemporary cars and compose powertrain, safety and comfort subsystems. The responsibility of these ECUs concerns all the subsystems. As usually each functionality is implemented in a separate ECU, their number in a car sometimes reaches even 100 [1]. This approach is not scalable, hence more effort has been put recently in the development of bus-based multi-core ECUs [2] or even ECUs whose multiple cores are connected with on-chip networks [3], capable of realising a number of ever more sophisticated functionalities in one chip. These functionalities are implemented in a form of so-called *runnables* which are atomic software components.

According to the AUTOSAR (AUTomotive Open System ARchitecture) standard [4], runnables are mapped to processing cores in a static manner, i.e. in a way utterly determined during the design-time. This approach is more predictable than dynamic (i.e. run-time) resource allocation but usually leads to underutilised resources as the underlying chips have to guarantee timing execution even for the worst-case scenario, as the runnables in automotive systems are usually bounded by hard real-time constraints [5]. Typically, worst-case execution time (WCET) of a runnable is much higher than the average execution time [6]. Hence some techniques decreasing the gap between WCET and the average execution time are desired. In this paper, we exploit the modal nature of automotive applications, i.e. the fact that runnables' execution time distributions vary depending on the overall system states, referred to as *modes*. Then the resource application process may be performed for each mode (almost) independently, providing different (lower) WCETs for runnables in the majority of modes in comparison with their counterparts when the modal nature is not considered.

To illustrate the concept of modes in an ECU, some characteristics of a gasoline engine control unit named DemoCar is presented in Fig. 1 (the idea of this picture is based on [7]). They are measured from inserting a key into ignition until reaching its full power. Five consecutive modes have been presented together with the corresponding throttle, revolutions per minute (RPM) and acceleration pedal positions. *PowerUp* is the starting mode which is present just after the key being inserted into the ignition. Then the *Stalled* mode follows, in which the throttle is still not open. The engine starts in the *Cranking* mode, hence the number of RPM increases reaching the idle RPM level. Then the mode is switched to *Idle*, which remains until the driver pushes the accelerator pedal. This switches the current mode into *Drive*, in which the throttle is wide open and the number of RPM is larger than the idle RPM level.

It has been stressed in [7] that execution times of particular runnables may differ significantly for various modes of an ECU and thus applying different mappings for each operating mode may be beneficial. This way a lower number of cores could be needed than that of the corresponding system not considering operating modes. This observation is true also for DemoCar. We can illustrate it with two runnables: *CylNumObserver* and *InstructionsDeviation*, whose numbers of instructions to be executed during one runnables' occurrence in the best- and worst-case scenarios are given in Table 1 (although this table is a small subset of both states and runnables, for the sake of this example we assume that there are no more states or runnables). From this table, it follows that the largest number of operations to be executed during a single occurrence (and hence the longest execution time) of runnable *CylNumObserver* is 543, and of *InstructionsDeviation* - 5921. Thus to guarantee the schedulability of these runnables in the worst-case scenario these extreme values have to be assumed, totalling 6464 operations. However, these two extreme numbers of operations cannot occur simultaneously, as the engine is either in mode *PowerDown* or *PowerUp*. In the former, the maximum number of operations during one runnable execution totals 1464, whereas in the latter it is 6266. So we can assume that even in the worst-case scenario the number of operations to be executed does not exceed 6266. The difference between the operation numbers in these modes suggests that less resources may be needed in *PowerDown* than in *PowerUp*, which arises the opportunity for some energy savings. This effect is even more visible when all runnables and modes are considered, which is shown later in this paper.

As stated above, there is a tempting possibility of performing resource allocation and schedulability analysis for each mode independently. Such allocation will be feasible as long as the system remains in a particular mode and presumably would require less resources and dissipate less energy. However, during mode change, the contexts of runnables that are executed on different cores in two subsequent modes need to be migrated from one core to another and the time of this migration shall be bounded, as hard real-time systems are considered in this paper. Therefore, the worst case switching time has to be assumed to provide the timing guarantees [8]. To migrate all involved runnables' contexts during a required interval, some additional requirements for the available communication bandwidth can be imposed. Thus the process of mode switching (e.g. from *PowerUp* to *Drive*) usually incurs overhead both in execution time and dissipated energy, if an allocation needs to be changed. This overhead needs to be taken into account during schedulability analysis to decide whether an altered resource allocation for a particular mode is beneficial for the whole system or not.

In the example above it has been demonstrated that the difference in the number of operations to be executed in various modes may be significant (in this particular case it has been more than 76%). It means that in some particular modes less computational resources are required to execute all runnables before their deadlines. These extra computational cores can be either switched off, or transferred into a more energy saving state if they support Dynamic Voltage and Frequency Scaling (DVFS) technique. DVFS is universally popular in CMOS circuits [9], in particular following the Advanced Configuration and Power Interface (ACPI) open standard. In this standard, several energy states, named P-states, are introduced. In the highest P-state, P0, a processing core works with the highest voltage and frequency level, but offers the best performance. In P1 and other modes, the core works slower, but dissipates less energy, as dynamic (or switching) power P is proportional to the square of core supply voltage V and its clock frequency f, $P \propto fV^2$. 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.

The DVFS technique seems to be particularly applicable to hard real-time systems, as in these systems there are 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 in order to lower energy dissipation if all the deadlines are satisfied.

Contribution: In this paper, we consider various modes of automotive applications to be mapped into a NoC-based multi-core system. We determine a quasi-optimal allocation for each mode by employing a genetic algorithm based approach. The genetic algorithm is a well-known metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms. It is commonly used to generate high-quality solutions to optimization and search problems by relying on bio-inspired operators such as mutation, crossover and selection [10]. Reference [10] details more about genetic algorithms. This approach performs optimization for energy dissipation and migration cost in terms of the context length of the transferred 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. A trade-off between the amount of transferred data during a mode change and the energy dissipation in the following state is investigated.

A preliminary version of this work has been published in [11]. This paper extends the work in [11] by including more detailed explanations and experimental results.

The proposed approach can be applied to any hard real-time systems, where different operating modes can be identified, and automotive systems in particular.

This paper is organised as follows. In the next section, the state-of-the-art solutions are reviewed. Then, the adopted application and platform models together with the problem formulation are described. In the following section, the steps of the proposed design flow are presented. Then, they are experimentally evaluated using two electronic control units from Robert Bosch GmbH. The paper is finalised with concluding remarks.

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. Traditionally, the research focused on single-processor systems [12] or, more recently, homogeneous bus-based multi-cores [13]. As the contemporary microcontrollers dedicated to the automotive industry, such as Infineon TriCore, follow these architecture principles, the schedulability analysis presented in [14] may be directly applied to them when modal applications are considered.

Recently, Network on Chip (NoC) has been proposed as a base for integrated system architecture for automotive electronic systems in [15]. The authors of that paper argued that the proposed architecture provided the required composability level and error containment to integrate multiple functions on a single ECU. In [3], the authors discussed the major benefits of using NoCs in the automotive domain, such as complexity reduction, reduction of resource requirements, increased dependability, legacy reuse and economic benefits. In [16], it was shown how NoCs can handle delay faults or process variations in automotive applications. The increased reliability was demonstrated in a scenario when processing cores become faulty one after another or in a case of a single link or router fault. This result was achieved by applying a dependable routing algorithm and dependable task execution. Additionally, the underlying processing cores were efficiently used for load balancing. Considering all these benefits, a NoC architecture may be expected to be applied in the automotive industry in foreseeable future and thus it has been also used in the solution proposed in this paper.

Since the number of possible scenarios in NoC-based multi-cores is typically prohibitively high [17], a number of research activities aims at developing design-time (off-line) heuristics to reduce the number of operating points in the design space exploration (DSE) process [17]. The 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 [7].

Two different mapping approaches are proposed in [18], 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. [19] 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 [20]. An FSM is an abstract machine that can be in exactly one of a finite number of states at any given time. The FSM can change from one state to another in response to some external inputs; the change from one state to another is called a transition. An FSM is defined by a list of its states, its initial state, and the conditions for each transition. This facilitates to identify all the allowed modes, represented as states in the FSM, and the transitions between them and to check the cost of mode switchings. In [21], 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 by a low number of the existing modes, obtained due to the clustering, and thus relatively rare switching. In [22], 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 minimise the amount of data to be migrated instead.

To guarantee hard real-time constraints during task migration, a methodology is proposed in [23]. 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 [7], 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 [24], 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 individuals. 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 dissipated in the future mode. Also, different modes in the DemoCar example have been identified.

The close observation of literature survey indicates that designing NoC-based 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 has been no proposal of any other method that jointly addresses the problems of (i) guaranteeing no hard deadline violation during mode switching, (ii) performing schedulability analysis for NoC-based multi-core systems, (iii) finding a trade-off between migration data amount and energy dissipation.

A preliminary version of this work is published in [11], which has been extended with more detailed explanations and results.

System model

In this paper, we investigate ways to determine whether an automotive modal application executing and communicating over a specific Network on Chip can meet all its hard deadlines. Therefore, we need a system model that covers the modal application as well as the NoC-based platform and its configurations.

Application model

In this work we assume that the application model is consistent with the AUTOSAR standard [4]. Each application (or taskset) Γ includes a set of p tasks with hard real-time constraints which can be represented with a vector $\Gamma = [\tau_1, \ldots, \tau_p]$. As this paper concerns the automotive domain, these tasks are referred to as *runnables*. The more general term *task* will only be used in reference to other application domains.

The taskset's properties depend on the current mode μ of the application. The runnables are periodic and each occurrence of a runnable is named a *job*. The taskset is known in advance, including the WCET of each runnable in every mode μ, $\mu = \{1, \ldots, m\}$, its period, priority and its relative deadline equal to this period. Runnables are atomic schedulable units communicating with 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 NoC nodes as the runnables that read these labels. Consequently, if more than one runnable mapped to different cores read from the same label, its content is to be replicated to all the NoC nodes with these cores and the writer should update the label value at all these locations. It means that the writer is aware of all its readers and knows their locations in all the possible modes.

Similarly to [23], we split a runnable's context into two parts: i) invariant, which is not modified at run-time, 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.

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

Platform model

The hardware platform assumed in this paper is a mesh Network on Chip (NoC). The rationale behind this choice is due to its several benefits as presented in the Related works section. The considered NoCs are assumed to have $x \cdot y$ nodes with cores, $\Pi = \{\pi_{1,1}, \ldots, \pi_{x,y}\}$ and local memories, and the same number of routers $\Psi = \{\psi_{1,1}, \ldots, \psi_{x,y}\}$, 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 $\pi_{1,2}$ to appropriate sink $\pi_{3,1}$ we need such link resources allocated simultaneously: $\pi_{1,2} - \psi_{1,2}$, $\psi_{1,2} - \psi_{2,2}$, $\psi_{2,2} - \psi_{3,2}$, $\psi_{3,2} - \psi_{3,1}$, $\psi_{3,1} - \pi_{3,1}$.

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. The data between two nodes is transmitted in packets. Each packet is comprised of a header, which includes the necessary control fields such as the destination address, and a payload with an actual intended data. 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. If a source and a target cores are not adjacent or if any contention exists among the data transfers, the worst-case transfer time is determined using the algebraic model described in [25]. Timing constants for packet latencies while traversing one router and one link are given and denoted as d_R and d_L, respectively. The priority of data transfer packets is 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.

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 the allowed mode changes and energy dissipated by the platform are minimised.

It might be possible that by increasing the amount of data to be migrated during a given mode change, the total energy consumption might be minimised because of a better mapping in the following mode. Therefore, a trade-off between these two objectives might exist and it should be illustrated with a Pareto frontier of points representing different energy consumption and amount of data to be migrated.

During mode switching, the taskset should be still schedulable despite the additional network traffic generated by the runnable migrations. The neighbouring modes with similar runnables' execution time can be clustered. This way, the number of modes is lower as clusters are used to group a set of modes into a single mode. Such reduction in the number of the existing modes can decrease the frequency of the runnable migrations, which is explained later in this paper.

The deadlines for mode switching time between each neighbouring pair of modes must not be violated.

Proposed approach

In this section, the steps of the proposed design flow are described. Since it has been assumed that the tasksets of the considered applications 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 is being changed, the current runnable and label mapping might need to be switched. The mappings have been identified at the design-time while trying to minimise the amount of data to be migrated during the static mapping for both initial and non-initial modes. Schedulability analysis 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 should be clustered, or the network bandwidth is to be increased.

In the remaining part of this section, we firstly explain the steps performed off-line, followed by the description of the dynamic stage.

Mode detection / clustering

During analysis of a modal system it may happen that runnables executed in neighbouring modes (i.e. the modes connected in the FSM) have similar WCETs and resource utilisations. In such case, there is little benefit in determining two separate mappings for these modes and migrating the runnables' contexts during transitions between these modes. It would be more reasonable to cluster these states and treat them as a single mode in the further steps of the proposed approach.

Similarly, some transitions between modes may have to be done immediately, whereas others can be less time tight. If runnable contexts' migration has to be performed quickly, for example between two consecutive runnable occurrences (jobs), the bandwidth needed to transfer the appropriate amount of data in that time may be unreasonably wide. In such situation, clustering of these modes shall also be considered.

The proposed approach is agnostic with respect to the chosen clustering method. In our implementation, the popular k-mean algorithm has been applied, whose idea was given in [26] for the first time. The features used for clustering are the WCETs (or the numbers of operations to be executed in the worst-case scenario) of particular runnables. Each mode is represented as a point in a p-dimensional vector space, referred later to as feature space. In the k-mean algorithm, a number of clusters, k, and m points in the p-dimensional feature space are provided as inputs. The number of clusters represents the number of groups in which all the points in the feature space need to be partitioned by employing a clustering algorithm, for example, the k-mean clustering algorithm. An appropriate value of k is often evident due to the knowledge about the relations between m modes in the considered application. If this knowledge is limited, one of the numerous existing solutions can be used to determine the right value of k, e.g. [27].

Initially, the k first points are treated as single-element clusters, and the remaining $m - k$ points are assigned to the cluster with the nearest centroid based on the Euclidean distance. Then the centroid for each cluster is recomputed. These two stages, i.e. assigning points to the cluster with the nearest centroid and the centroid re-computation are repeated until convergence. A set of k clusters is returned as output. The modes grouped into one cluster are merged into one mode in which WCET for each runnable is equal to the maximal WCET for the particular runnable in any mode grouped in this cluster.

This concept may be illustrated with the following simple example. Let us have an application with $p = 2$ runnables $\Gamma = [\tau_1, \tau_2]$ in $m = 5$ distinctive modes. As there are only two runnables, the m points in a feature space have two dimensions, one corresponding to the WCET for τ_1 and the second one corresponding to the WCET for τ_2, and thus they can be shown on a plane. These feature points are presented as five circles in Fig. 5, where the OX and OY coordinates represent the WCETs (in ms) for τ_1 and τ_2, respectively. The number of clusters has been set to $k = 3$ and thus three centroids have been found, shown in the figure with crosses. The lines in the figure divide the plane into segments that are closer to a certain centroid than to the remaining ones. The modes described with the feature points belonging to the same segment are merged into a single mode. For example, in the uppermost segment two feature points can be found: $[1, 5]$ and $[2, 6]$. After merging the corresponding modes, the WCET for the clustered mode equals $max(1, 2) = 2$ms for τ_1 and $max(5, 6) = 6$ms for τ_2.

Spanning tree construction

In the proposed approach, the FSM describing the modes is traversed starting from the initial mode and then the runnable migration corresponding to each traversed transition is analysed. In this traversal each mode should be analysed exactly once as only one mapping is assigned to one mode. If a particular mode can be reached from a number of different modes, the most probable transition shall be chosen. Hence the FSM describing mode changes should include weights denoting state transition probabilities. These probabilities can be given or determined during a simulation of the modal system. Then the FSM is treated as a weighted directed connected graph $G(V, E)$, where V is the set of vertices $\{v_1, v_2, \ldots\}$ and E denotes the set of directed edges. We firstly convert this graph into its undirected counterpart, $G(V, E')$, where set E' includes an edge (v_k, v_l) if and only if $(v_k, v_l) \in E \vee (v_l, v_k) \in E$. The weight of edge $(v_k, v_l) \in E'$, $\omega(v_k, v_l)$ is equal to the sum of weights of edges $(v_k, v_l) \in E$ and $(v_l, v_k) \in E$.

We use an algorithm for undirected graphs, as we take into consideration the probability of mode switching in both directions, i.e. the sum of these probabilities for two directed edges connecting these states in the related FSM (the weights cannot be thus treated as probabilities in the undirected graph as they may be higher than 1).

To guarantee a single analysis of each mode while following the most probable paths, a maximum spanning tree can be constructed. We recollect that a spanning tree of a graph $G(V, E')$ is its subgraph $T(V, E'')$ which is connected and whose number of edges is equal to the number of vertices minus 1, $|E''| = |V| - 1$. If $\mathcal{T}$ denotes the set of all spanning trees of G, a maximum spanning tree $T_{max}(V, E''_{max})$

of G is a spanning tree if and only if:

$$\underset{T(V,E'')\in\mathcal{T}}{\forall} \sum_{(v,z)\in E''_{max}} \omega(v,z) \geq \sum_{(v,z)\in E''} \omega(v,z),$$

where $\omega(v,z)$ is the weight value assigned to the edge from a vertex v to z. A maximum spanning tree can be constructed in time $O(|E'|log|V|)$ using the classic Prim–Jarník's algorithm [28].

According to this greedy algorithm, a tree is initialised with an arbitrary vertex. In our implementation, we select the vertex corresponding to the initial state in the FSM. Then, in each step, one vertex is chosen and added to the tree. This selected vertex is not yet in the tree and is connected with any tree vertex with an edge having the largest weight. This operation is repeated until all vertices are added to the tree.

Let us illustrate this idea with a simple example, an FSM with three states, A, B and C, presented in Fig. 6a. The corresponding undirected weighted graph is shown in Fig. 6b. Vertex A is selected as the first vertex of the spanning tree (Fig. 6c). Two vertices are adjacent to the spanning tree, B and C, which is shown in the figure with dashed lines. In the next step, vertex B is added to the spanning tree, as it is connected with vertex A with an edge with the largest weight, $\omega(A,B) = 0.9$ (Fig. 6d). Vertex C is adjacent to the tree and can be connected to vertex A or B. Then, in the third step, vertex C is connected to vertex B as this edge has a larger weight, $\omega(B,C) = 1.4 > \omega(A,C) = 0.7$. Since all the vertices have been added to the spanning tree, the Prim–Jarník's algorithm is completed. The maximum spanning tree is presented in Fig. 6e.

Notice that the operation performed in this step neither influences the application behaviour nor limits the possible mode transitions. It only makes the least frequent transitions not optimized during the further stages.

Static mapping

In the proposed approach the algorithms for resource allocation in the initial and the remaining modes vary, and thus they are presented separately in the following two subsections.

Initial mode

Algorithm 1 presents a pseudo-code of a genetic algorithm that can be used to identify a mapping for the initial 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. 7. The first p genes indicate the target cores for p runnables and the remaining $|\Psi|$ genes (for a mesh NoC $|\Psi| = x \cdot y$, where x and y are the mesh dimensions) specify the P-states of the consecutive cores.

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

```
inputs  : Workload $\Gamma$;
          Resource set $\Pi$;
outputs : Runnable mapping; Core P-states;
1  Generate an initial random population of runnable 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      Group together the individuals with the same number of deadline violations;
6      Sort the groups by increasing number of deadline violations;
7      Sort individuals in each group with respect to 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
```

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

Step 1. Initial population generation (line 1). An arbitrary number of random individuals (runnable mappings 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 a lower number of deadline violations are always preferred, whereas among the individuals from one group the one with the lower dissipated energy is to be chosen (line 8). The computation of the tournament outcome is characterised with low overhead due to the appropriate ordering of the groups and individuals in each group performed earlier. The individuals winning the tournament are then combined 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.

For example, assume that a population is comprised of four individuals, i_1, i_2, i_3 and i_4. The evaluation of these individuals made in line 3 shows that for individuals i_1 and i_3 as many as 2 deadlines are missed, whereas the mappings for i_2 and i_4 are schedulable considering the P-state assignments encoded in these individuals. According to the energy dissipation evaluated in line 4, i_1 dissipates the lowest energy, followed by i_2, i_3 and i_4 (in this order).

Since i_1 and i_3 violate the same number of deadlines, they are joined together in one group, $Group_1$, according to line 5. Individuals i_2 and i_4 are grouped together in $Group_2$ as they do not violate any deadline. In each group, the individuals are sorted with respect to the dissipated energy in the ascending order (line 7).

In the tournament selection performed in line 8, each individual from $Group_2$ wins over any individual from $Group_1$, as the number of violated deadlines is the more important criterion. So, for example, i_4 beats i_1 despite dissipating more energy. If a tournament is performed for two individuals from the same group, i.e. violating the same number of deadlines, the individual characterised with a lower energy dissipation is the winner. For example, if both individuals from $Group_1$ enter the tournament, i_1 becomes the winner.

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 dissipated in the next mode is much lower than the migration energy. Thus there could be some trade-off between migration data (or time) and energy dissipation in the next mode. It is a role of a designer to choose a proper solution from the Pareto frontier.

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 Γ (taskset) and can be substituted with any element of set Π (processing cores).

To perform optimization for the 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 $(\tau_1, \ldots, \tau_p)$ is migrated, including the labels to be read or written.

Let $\mathcal{M}_\alpha$ and $\mathcal{M}_\beta$ be sets of mappings (i.e. sets of M vectors) that are fully schedulable in a given system in modes α and β, respectively. The elements of the difference vector $\mathrm{D}_{\mathrm{M}_\alpha,\mathrm{M}_\beta} = [d_{\tau_1}, \ldots, d_{\tau_p}]$ 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 is allocated to different cores in mappings $\mathrm{M}_\alpha \in \mathcal{M}_\alpha$ and $\mathrm{M}_\beta \in \mathcal{M}_\beta$, and 0 otherwise:

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

where $\mathrm{M}_{\alpha,\delta}$ and $\mathrm{M}_{\beta,\delta}$ 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

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

For example, we consider a taskset with three runnables $\Gamma = [\tau_1, \tau_2, \tau_3]$. The elements of vector $\mathrm{W} = [100, 200, 150]$ describe the context lengths (in bytes) of τ_1, τ_2 and τ_3, respectively. Let, there is one mapping in mode α, $\mathcal{M}_\alpha = \{\mathrm{M}_{\alpha 1}\}$ and two mappings in mode β, $\mathcal{M}_\beta = \{\mathrm{M}_{\beta 1}, \mathrm{M}_{\beta 2}\}$, where $\mathrm{M}_{\alpha 1} = [\pi_1, \pi_1, \pi_2]$, $\mathrm{M}_{\beta 1} = [\pi_1, \pi_2, \pi_2]$ and $\mathrm{M}_{\beta 2} = [\pi_2, \pi_1, \pi_1]$. Thus the corresponding difference vectors equal $\mathrm{D}_{\mathrm{M}_{\alpha 1},\mathrm{M}_{\beta 1}} = [0, 1, 0]$ and $\mathrm{D}_{\mathrm{M}_{\alpha 1},\mathrm{M}_{\beta 2}} = [1, 0, 1]$. The migration costs between these mappings are $c_{\mathrm{M}_{\alpha 1},\mathrm{M}_{\beta 1}} = \mathrm{D}_{\mathrm{M}_{\alpha 1},\mathrm{M}_{\beta 1}} \cdot \mathrm{W}^{\mathrm{T}} = 200$ and $c_{\mathrm{M}_{\alpha 1},\mathrm{M}_{\beta 2}} = \mathrm{D}_{\mathrm{M}_{\alpha 1},\mathrm{M}_{\beta 2}} \cdot \mathrm{W}^{\mathrm{T}} =$

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   $\alpha \leftarrow$ the state of ST corresponding to the initial state of FSM;
2   $\mathcal{M}_\alpha \leftarrow$ a set of schedulable mappings in mode $\alpha$;
3   $M_\alpha \leftarrow$ the mapping in $\mathcal{M}_\alpha$ that dissipates the lowest amount of energy;
4   forall the $\beta$ being a direct successor of $\alpha$ in $ST$ do
5   |   $M_\beta \leftarrow$ FindMappingMin($\alpha$, $\beta$, $M_\alpha$);
    end

     procedure FindMappingMin($\eta$, $\kappa$, $M_\eta$)
1.1  $\mathcal{M}_\kappa \leftarrow$ a Pareto frontier of schedulable mappings in $\kappa$ minimizing criterion Equation (2)
    and energy dissipation in mode $\kappa$ using $W$;
1.2  $M_\kappa \leftarrow$ the mapping in $\mathcal{M}_\kappa$ selected with respect to design priorities;
1.3  forall the $\iota$ being a direct successor of $\kappa$ in $ST$ do
1.4  |   $M_\iota \leftarrow$ FindMappingMin($\kappa$, $\iota$, $M_\kappa$);
    end
1.5  return $M_\kappa$;
```

250 bytes. If minimisation of the migrated data size is the only criterion, mapping $M_{\beta 2}$ shall be chosen for mode β.

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 the Finite State Machine describing transitions between modes, a maximum spanning tree (ST) is to be built, as described earlier. 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 dissipation is selected (line 3). Then for each direct successor of the ST vertex corresponding to the 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 vertex is found using two criteria: i) minimal migration cost criterion represented by 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 vertex 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 $\mathcal{M}_\alpha$ 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.

If Algorithm 2 is applied to the example spanning tree presented in Fig. 6e, mode A is substituted to α as it corresponds with the initial state of the FSM shown in Fig. 6a. Then the mapping M_α that is schedulable and dissipates the lowest amount of energy is determined using Algorithm 1. As there is only one direct successor of mode A in the spanning tree, the *FindMappingMin* procedure is executed for mode B. In this procedure, a Pareto frontier between schedulable mappings in mode B minimizing energy dissipation in B and the amount of data to be transferred during mode switching from A to B is determined. After selecting the appropriate mapping using the assumed design priorities, procedure *FindMappingMin* is executed again for mode C, the only successor of mode B in the spanning tree.

Schedulability analysis

The proposed runnable mapping technique aims to benefit from the modal nature of applications, but it also possesses 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 [29]. However, the instant of transition between the modes requires special attention, as additional migration-related traffic appears including the whole contexts of the runnables and labels to be migrated. To guarantee the taskset schedulability during migration, we propose to treat a migration process as any other asynchronous process in the typical 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 runnable migration. If there is no pending migration, the server simply holds its capacity.

The dynamic (i.e. changeable) part of the context shall be migrated at once using the last job 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 hyperperiod, where hyperperiod is the least common multiple of all runnables' periods) and, consequently, the network bandwidth, as it must be then wide enough to guarantee migration of the dynamic part before the next runnable's job 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 runnables, as described for example in [30], shall be considered. More details regarding the applied schedulability analysis scheme in the proposed approach are provided in [24].

On-line steps

In the proposed approach, three steps are performed on-line: *Detection of current mode*, *Mapping switching* and *Changing voltage/frequency levels of cores.*

In all the ECUs known to the authors of this paper, the system modes are defined explicitly and there is a possibility of determining the current mode by observing some system model variables, similarly to [7]. (For example, in DemoCar such variable is named *_sm* and is stored in runnable *OperatingModeSWCRunnableEntity*.)

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 contexts of these runnables are migrated. 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 jobs that have to be used during the whole migration process, and has the dynamic (volatile) portion of the context identified. This part of the context is to be transmitted in a single job of the periodic server, just after the last execution of the runnable at its old location. After migrating the dynamic part of the runnable's context, it is removed from the earlier (migration source) core.

Simultaneously, the same agent can receive migration data from other agents in the network. When the precomputed (during the design-time) number of hyperperiods elapses, the contexts of these runnables are fully migrated and are ready to be executed on the migration target core.

Before the first execution of a runnable in a new mode, the agent switches the P-state of the processing core to the value determined during the static analysis, described earlier in this paper.

The details of the agent depend on the underlying operating system. Regardless of its implementation, *Detection of current mode* shall be characterised by low computational complexity and thus shall impose low overhead for the system during run-time. The number of the hyperperiods required for performing runnable 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 constants for packet latencies while traversing one router and one link (d_R and d_L). This dependency will be explored in the following section.

Experimental results

As examples, we consider two industrial applications, an engine control management (ECM) and a lightweight engine control system named DemoCar. In this section, we follow the stage order presented in Fig. 4. We begin with mode clustering and construction of the spanning tree, followed by runnables' mapping and determining of the required network bandwidth.

The energy dissipation has been determined using the technique described in [31]. The average values obtained during a series of simulations are following. During 1s, a processing core dissipates 1.66E+4μJ when idle and 2.99E+4μJ when busy. An idle link dissipates 1.43E+3μJ whereas a link transporting a package dissipates 3.69E+3μJ during 1s assuming 16-bit wide data connection links between cores. The core energy has been scaled using relation $P \propto fV^2$ for a set of six P-states, from P0, where the maximum voltage/frequency level has been assigned, to P5.

For the considered applications, next, we describe the results of i) mode clustering and spanning tree construction, ii) mappings exploration for initial and non-initial modes, and iii) energy and data migration trade-off influence on NoC bandwidth.

Mode clustering and spanning tree construction

The first of the considered applications, ECM, is comprised of three modes, in which runnables have different best and worst case number of operations to be executed, as presented in Table 2. The transitions between all these modes are possible, as shown in Figure 8. The hyperperiod for this application is equal to 200ms.

From Table 2 it follows that two modes, *Mode 1* and *Mode 2*, only differ in absence of three runnables (*Runnable_02*, *Runnable_03* and *Runnable_06*) in the latter. The remaining runnables have exactly the same number of operations to be executed in the best and worst case scenarios. It may be then beneficial to decrease the number of modes to $k = 2$ by clustering *Mode 1* and *Mode 2*. The maximum numbers of operations for all 10 runnables have been treated as features, so for *Mode 1*, *Mode 2* and *Mode 3* their corresponding points in the feature space are equal to [363, 2, 222, 600000, 60000, 16700, 100000, 200000, 200000, 200000], [363, 0, 0, 600000, 60000, 0, 100000, 200000, 200000, 200000] and [363, 2, 222, 121, 209, 167, 80, 67, 25, 171], respectively. The k-mean clustering algorithm has found the following two centroids: [363, 1, 111, 600000, 60000, 8350, 100000, 200000, 200000, 200000] and [363, 2, 222, 121, 209, 167, 80, 67, 25, 171]. Modes *Mode 1* and *Mode 2* are closer to the first centroid, so they are clustered in a single mode, *Cluster 1*, as shown in Fig. 9. Notice that the Euclidean distances between the two points in the feature space corresponding to *Mode 1* and *Mode 2* and the centroid are relatively low, which means that these two modes do not differ significantly and thus merging them into one mode can be beneficial. As the maximal numbers of operations to be executed in each runnable in *Mode 1* are higher than or equal to the corresponding numbers in *Mode 2*, the former are assumed to be used for the whole cluster. For such simple FSM with only two states, creating the maximal spanning tree is rather trivial. The only edge is the one connecting the initial mode *Cluster 1* with *Mode 3*.

Our second analysed case, a gasoline engine control unit named DemoCar, is a larger application as it consists of 18 runnables and 61 labels.

Its flow graph has been presented in [32] together with a detailed description. In Fig. 2, seven 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 jobs of their runnables, which is upperbounded with 5ms for 9 runnables. Since performing runnable migration during such short time window would require a bandwidth of considerable size, these modes have been clustered into *Cluster1*. For 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. 10. The probabilities of mode switching have been shown above the arrows. The maximum spanning tree, constructed with the Prim–Jarník's algorithm, is presented in Fig. 11.

Mappings exploration for initial and non-initial modes

For the ECM application, the genetic algorithm presented in Algorithm 1 has been executed for the initial mode, *Cluster 1*. This algorithm has been configured to generate 100 generations of 20 individuals each. The size of the NoC has been initially set to 1x1 and the flit size has been fixed to 16 bits. For this architecture, the genetic algorithm failed to find any schedulable mapping. As many as 320 jobs out of 409 present in a hyperperiod have been executed after their deadlines in the best solution found. Not surprisingly, this solution has been reported for P-State P0. In this state, the core is characterised by the best performance while dissipating the highest amount of energy, equal to 10333.4μJ per hyperperiod.

The NoC size has been then enlarged to 2x1 and the flit size has not been altered. For this architecture, a number of schedulable mappings has been found. The schedulable mapping with the lowest dissipated energy, equal to 11912.6μJ per hyperperiod, assigned P-States P1 and P3 to the first and the second processing core, respectively. The first core has been assigned with 7 runnables: *Runnable_02*, *Runnable_03*, *Runnable_04*, *Runnable_06*, *Runnable_07*, *Runnable_09*, *Runnable_10*, whereas the remaining 3 runnables have been assigned to the second core.

The algorithm presented in Algorithm 2 has been applied to find a schedulable mapping in the second mode, *Mode 3*. As the total number of operations to be executed are much lower than in the initial mode (see Table 2), the mapping has been performed to only one core. The second core has been decided to be switched off in this mode.

Consequently, only two mappings were possible: all runnables from the first core can be migrated to the second core and vice versa. The number of possible solutions is larger though, as different P-states can be applied to the chosen core. One solution dominated the remaining ones in both the criteria. In this mapping, 3330.43μJ are dissipated per hyperperiod and 267725 bytes have to be migrated. The most energy efficient P-state, named P5, has been sufficient to timely execute all the runnables.

The same procedure has been applied to the DemoCar application. For the *PowerUp* (initial) mode to be executed on a NoC-based multi-core system, we estimate the dissipated energy and number of violated deadlines during one hyperperiod 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) to be 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 again to 16 bits. The timing characteristics, energy dissipation per hyperperiod and the number of idle resources for different modes are summarised in Table 3 and commented below.

The genetic algorithm has been executed to perform assignment of the runnables 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 in P-State P0. The energy dissipated in this mode equals to 3093.01μJ per hyperperiod (100 ms). Thus, thanks to the modal approach, one can switch off 75% of the cores while the application is in the *Cluster1* mode.

As stated above, for the *PowerUp* mode, schedulable mappings have been found even if three of the four 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 cores has been identified. A Pareto frontier using two criteria, minimal amount of data to be migrated and minimal energy dissipated in the next mode, has been constructed and drawn in Fig. 12. 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 the minimal energy dissipation is crucial for the system, the solution leading to dissipation of 9719.45μJ (in the next mode) should be chosen. Then, using the same priority, the mapping in the *Cluster2* mode would dissipate 5909.37μJ per hyperperiod.

Energy and data migration trade-off influence on NoC bandwidth

As it has been shown above, the ECM application has been mapped to a low number of processing cores (one or two, depending on the current mode) so that the underlying NoC lacked any multi-hop links. For this application, one solution dominated the others and thus no Pareto curve has been built, as explained in the previous subsection. In this dominating solution, 267725 bytes have to be migrated between the NoC nodes when the current mode is being changed. This has to be added to 2660 bytes that are sent and received by these cores during each hyperperiod due to the runnable execution. Such amount of data can be migrated using a periodic server during one hyperperiod (200ms) as long as the router (d_R) and link (d_L) latencies do not exceed 500ns and 200ns, respectively.

In the case of the DemoCar application, we have evaluated the number of hyperperiods required to migrate runnables from *PowerUp* to *Cluster1*, depending on constants d_R and d_L, and presented them in Table 4. Two extreme solutions from the Pareto frontier illustrated in Fig. 12 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 100 and 50ns, respectively, for both the extreme solutions. However, when the routers and links have higher latency, the NoC needs a significant lower number of hyperperiods to migrate all the data in solution A.

Conclusions

An approach for runnable migration in a NoC-based multi-core system has been proposed as a way to decrease the number of cores needed for guaranteeing safe execution of a hard real-time software. Applying different voltage/frequency levels (P-states) to cores facilitates decreasing of energy dissipation even further. The proposed approach is comprised of steps to be performed statically (off-line) and during run-time (on-line). The approach has been illustrated with two industrial applications. In both of them, a Finite State Machine describing mode changes has been extracted from their code and transition 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 mode has been used for the remaining modes. 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. In the first application, in one mode a single processing core is sufficient to execute all the functionalities before their deadlines, whereas in the remaining modes two processing cores are required. In the case of the engine control unit, in a particular mode only quarter of the initial number of cores is used.

The proposed approach requires the 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.

Competing interests
The authors declare that they have no competing interests.

Author's contributions
Piotr Dziurzanski: the design flow stages, algorithms, using periodic server for data migration, proposal of the trade-off between the amount of migrated data and energy dissipation, software implementation of the described algorithms and performing experiments.
Amit Kumar Singh: general idea about hybrid approach, co-author of system model description, co-author of related work section.
Leandro Soares Indrusiak: general idea about multidimensional genetic algorithm used for schedulability-related problems, input on energy estimation macromodels for NoCs, input on dynamic task mapping heuristics for NoCs and co-author of related work section.

Funding
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 and the EPSRC through the project LowPowNoC - Evaluation and reduction of power dissipation in multicore systems based on Networks-on-Chip (Ref EP/J003662/1).

Acknowledgements
The authors would like to thank Björn Saballus and Devendra Rai from Robert Bosch GmbH for providing ECM and Democar ECUs.

References

1. Natale, M.D., Sangiovanni-Vincentelli, A.L.: Moving from federated to integrated architectures in automotive: The role of standards, methods and tools. Proceedings of the IEEE **98**(4), 603–620 (2010)
2. Fuhrman, T., Wang, S., Jersak, M., Richter, K.: On designing software architectures for next-generation multi-core ECUs. SAE International Journal of Passenger Cars - Electronic and Electrical Systems **8**(1), 115–123 (2015)
3. Obermaisser, R., Salloum, C.E., Huber, B., Kopetz, H.: From a federated to an integrated automotive architecture. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **28**(7), 956–965 (2009)
4. AUTOSAR development partnership: Autosar: Automotive open system architecture (2015)
5. Monot, A., Navet, N., Bavoux, B., Simonot-Lion, F.: Multisource software on multicore automotive ECUs - combining runnable sequencing with task scheduling. IEEE Transactions on Industrial Electronics **59**(10), 3934–3942 (2012)
6. Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., Mueller, F., Puaut, I., Puschner, P., Staschulat, J., Stenström, P.: The worst-case execution-time problem – overview of methods and survey of tools. ACM Trans. Embed. Comput. Syst. **7**(3), 36–13653 (2008)
7. Park, J., Harnisch, J., Deubzer, M., Jeong, K., Leteinturier, P., Suh, I.: 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. **7**(1), 133–140 (2014)
8. van Kampenhout, J.R.: Deterministic task transfer in network-on-chip based multi-core processors. Computer Engineering (18) (2011)
9. Quiñones, E., Abella, J., Cazorla, F.J., Valero, M.: Exploiting intra-task slack time of load operations for DVFS in hard real-time multi-core systems. SIGBED Rev. **8**(3), 32–35 (2011)
10. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: Nsga-ii. Trans. Evol. Comp **6**(2), 182–197 (2002)
11. Dziurzanski, P., Singh, A.K., Indrusiak, L.S.: Energy-aware resource allocation in multi-mode automotive applications with hard real-time constraints. In: IEEE 19th International Symposium on Real-Time Distributed Computing (ISORC), pp. 100–107 (2016)
12. Real, J., Crespo, A.: Mode change protocols for real-time systems: A survey and a new proposal. Real-Time Syst. **26**(2), 161–197 (2004)
13. Goossens, J., Richard, P.: Partitioned scheduling of multimode multiprocessor real-time systems with temporal isolation. In: Proceedings of the 21st International Conference on Real-Time Networks and Systems. RTNS '13, pp. 297–305. ACM, New York, NY, USA (2013)
14. Negrean, M., Klawitter, S., Ernst, R.: Timing analysis of multi-mode applications on AUTOSAR conform multi-core systems. In: 2013 Design, Automation Test in Europe Conference Exhibition (DATE), pp. 302–307 (2013)
15. Kopetz, H., Obermaisser, R., Salloum, C.E., Huber, B.: Automotive software development for a multi-core system-on-a-chip. In: Software Engineering for Automotive Systems, 2007. ICSE Workshops SEAS '07. Fourth International Workshop On, pp. 2–2 (2007)
16. Yoneda, T., Imai, M., Saito, H., Hanyu, T., Kise, K., Nakamura, Y.: An NoC-based evaluation platform for safety-critical automotive applications. In: 2014 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS), pp. 679–682 (2014)
17. Singh, A.K., Shafique, M., Kumar, A., Henkel, J.: Mapping on multi/many-core systems: Survey of current and emerging trends. In: Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE, pp. 1–10 (2013)
18. Schranzhofer, A., Chen, J.J., Thiele, L.: Dynamic power-aware mapping of applications onto heterogeneous MPSoC platforms. IEEE Transactions on Industrial Informatics **6**(4), 692–707 (2010)
19. Benini, L., Bertozzi, D., Milano, M.: Resource management policy handling multiple use-cases in MPSoC platforms using constraint programming. In: Proceedings of the 24th International Conference on Logic Programming. ICLP '08, pp. 470–484. Springer, Berlin, Heidelberg (2008)
20. Schor, L., Bacivarov, I., Rai, D., Yang, H., Kang, S.-H., Thiele, L.: Scenario-based design flow for mapping streaming applications onto on-chip many-core systems. In: Proceedings of the 2012 International Conference on Compilers, Architectures and Synthesis for Embedded Systems. CASES '12, pp. 71–80. ACM, New York, NY, USA (2012)
21. Gheorghita, S.V., Palkovic, M., Hamers, J., Vandecappelle, A., Mamagkakis, S., Basten, T., Eeckhout, L., Corporaal, H., Catthoor, F., Vandeputte, F., Bosschere, K.D.: System-scenario-based design of dynamic embedded systems. ACM Trans. Des. Autom. Electron. Syst. **14**(1), 3–1345 (2009)
22. Stuijk, S., Geilen, M., Theelen, B., Basten, T.: Scenario-aware dataflow: Modeling, analysis and implementation of dynamic applications. In: Embedded Computer Systems (SAMOS), 2011 International Conference On, pp. 404–411 (2011)
23. Munk, P., Saballus, B., Richling, J., Heiss, H.U.: Position paper: Real-time task migration on many-core processors. In: Architecture of Computing Systems. Proceedings, ARCS 2015 - The 28th International Conference On, pp. 1–4 (2015)
24. Dziurzanski, P., Singh, A.K., Indrusiak, L.S., Saballus, B.: Hard real-time guarantee of automotive applications during mode changes. In: Proceedings of the 23rd International Conference on Real Time and Networks Systems. RTNS '15, pp. 161–170. ACM, New York, NY, USA (2015)
25. Indrusiak, L.S., Dziurzanski, P.: An interval algebra for multiprocessor resource allocation. In: 2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, SAMOS 2015, Samos, Greece, July 19-23, 2015, pp. 165–172 (2015)
26. Steinhaus, H.: Sur la division des corps materiels en parties. Bull. Acad. Pol. Sci., Cl. III **4**, 801–804 (1957)

27. Sugar, C.A., James, G.M.: Finding the number of clusters in a dataset. Journal of the American Statistical Association **98**(463), 750–763 (2003)
28. Prim, R.C.: Shortest connection networks and some generalizations. Bell System Technical Journal **36**(6), 1389–1401 (1957)
29. Shi, Z., Burns, A.: Real-time communication analysis for on-chip networks with wormhole switching. In: Proceedings of the Second ACM/IEEE International Symposium on Networks-on-Chip. NOCS '08, pp. 161–170. IEEE Computer Society, Washington, DC, USA (2008)
30. Davis, R.I., Burns, A.: A survey of hard real-time scheduling for multiprocessor systems. ACM Comput. Surv. **43**(4), 35–13544 (2011)
31. Latif, K., Effiong, C., Gamatie, A., Sassatelli, G., Zordan, L., Ost, L., Dziurzanski, P., Indrusiak, L.: An integrated framework for model-based design and analysis of automotive multi-core system. In: Forum on Specification & Design Languages. FDL '15, Work-in-Progress Session (2015)
32. Dziurzanski, P., Singh, A.K., Indrusiak, L.S., Saballus, B.: Benchmarking, system design and case-studies for multi-core based embedded automotive systems. In: Proceedings of the 2nd International Workshop on Dynamic Resource Allocation and Management in Embedded, High Performance and Cloud Computing. DREAMCloud'16 (2016)

Figures

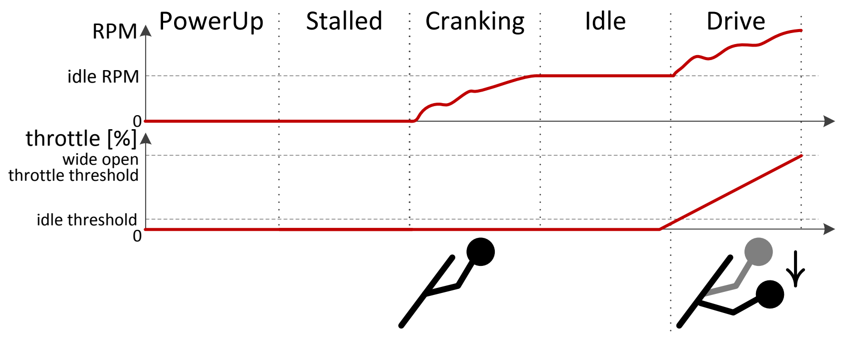

Figure 1 Modes in DemoCar

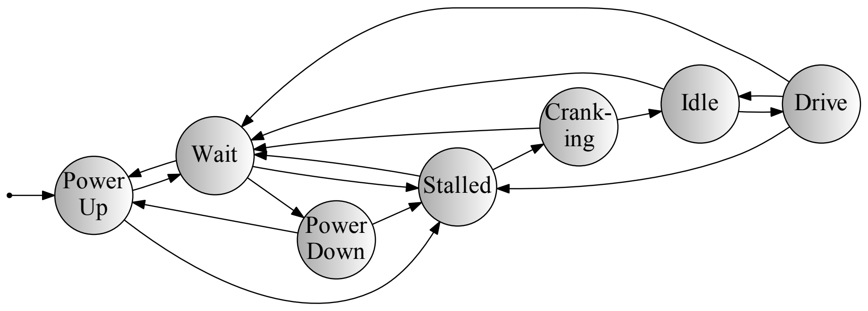

Figure 2 FSM describing modes in DemoCar

Tables

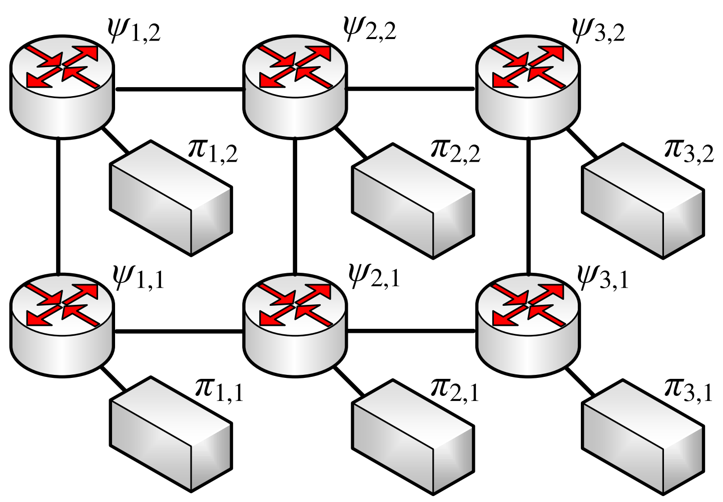

Figure 3 An example many-core system platform

Table 1 Numbers of operations for two runnables in two states in the best-case and worst-case scenarios

	Mode			
	PowerDown		PowerUp	
Runnable	Min. Op.	Max. Op.	Min. Op.	Max. Op.
CylNumObserver	245	543	134	345
InstructionsDeviation	728	921	3728	5921

Table 2 Numbers of operations for runnables in all three modes of the ECM application in the best-case and worst-case scenarios

	Mode 1		Mode 2		Mode 3	
Runnable	Min. Op.	Max. Op.	Min. Op.	Max. Op.	Min. Op.	Max. Op.
Runnable_01	12	363	12	363	12	363
Runnable_02	0	2	0	0	0	2
Runnable_03	29	222	0	0	29	222
Runnable_04	300000	600000	300000	600000	2	121
Runnable_05	55000	60000	55000	60000	5	209
Runnable_06	10000	16700	0	0	2	167
Runnable_07	20000	100000	20000	100000	2	80
Runnable_08	100000	200000	100000	200000	1	67
Runnable_09	150000	200000	150000	200000	5	25
Runnable_10	100000	200000	100000	200000	30	171

Table 3 Timing and energy characteristics of 2x2 mesh NoC executing DemoCar in different modes

Mode	# deadline violations	Energy dissipation per hyperperiod	#Idle cores	#Idle links
PowerUp	0	3093.01μJ	3	16
Cluster1	0	9719.45μJ	0	10
Cluster2	0	5909.37μJ	2	10

Table 4 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	50	1	1
200	100	1	2
400	100	1	2
500	200	2	4
800	400	3	7
1000	500	4	8

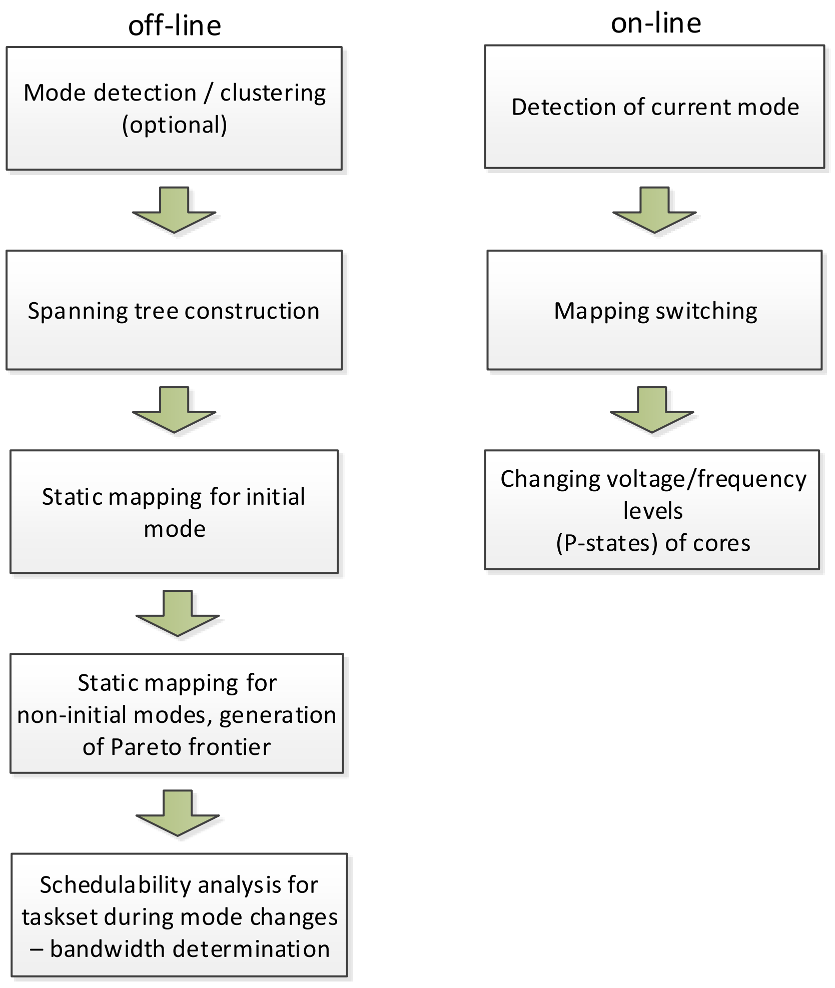
off-line
Mode detection / clustering (optional)
Spanning tree construction
Static mapping for initial mode
Static mapping for non-initial modes, generation of Pareto frontier
Schedulability analysis for taskset during mode changes – bandwidth determination
on-line
Detection of current mode
Mapping switching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

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

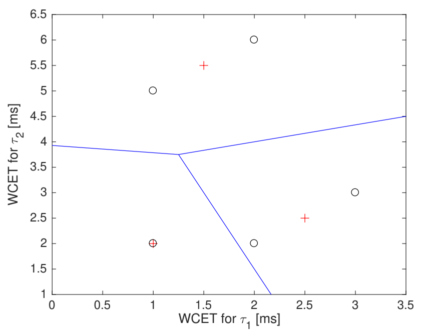

Figure 5 Example of five modes grouped into three clusters

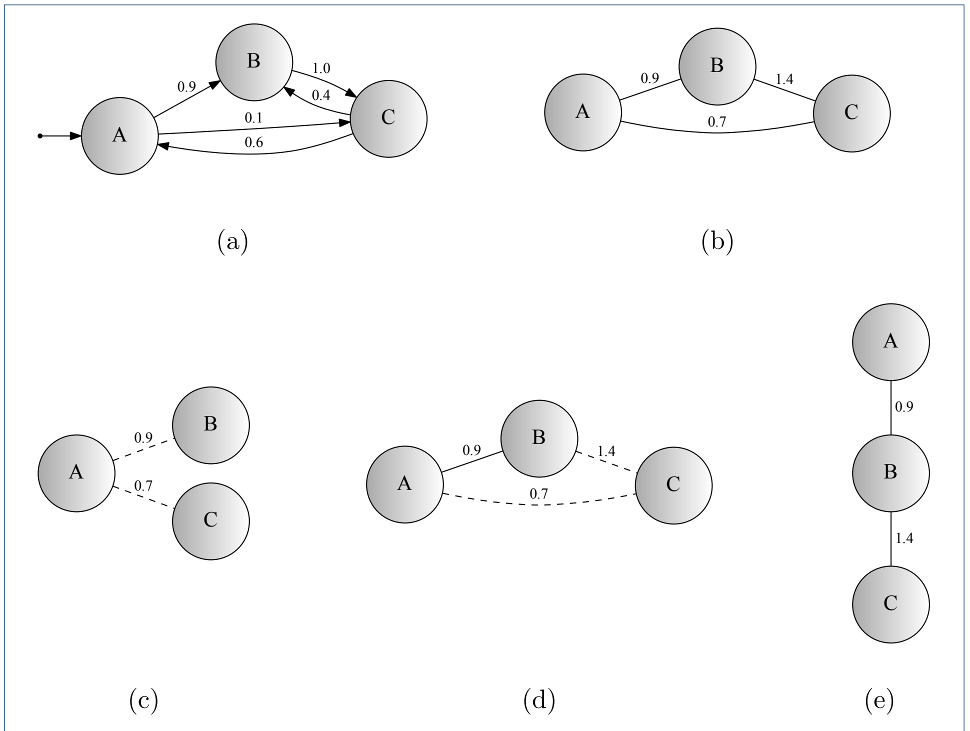

Figure 6 Example of the maximum spanning tree construction: FSM describing mode changes (a), the correponding undirectional graph (b), first (c), second (d) and third (e) step of the Prim–Jarník's algorithm

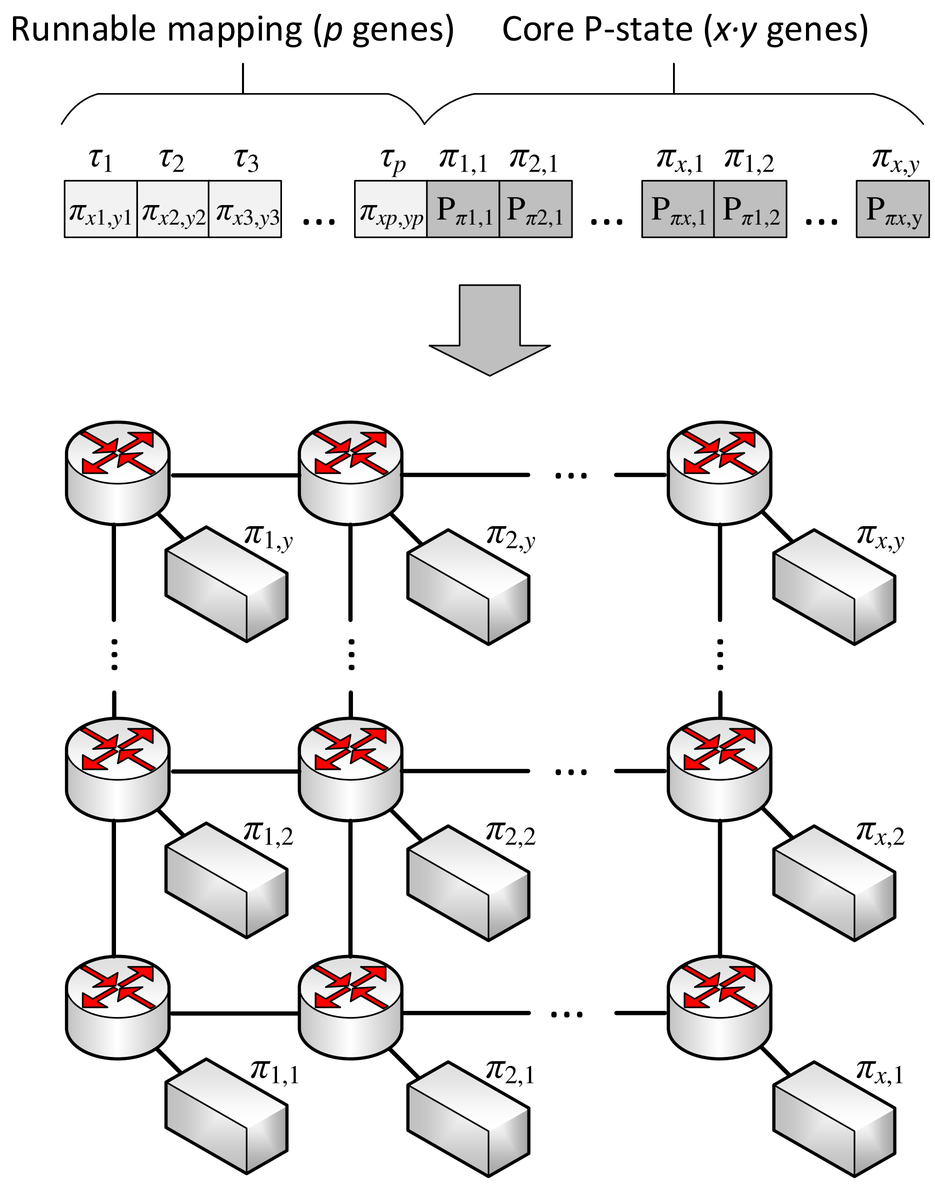
Runnable mapping (*p* genes)
Core P-state (*x·y* genes)
τ_1 τ_2 τ_3 τ_p $\pi_{1,1}$ $\pi_{2,1}$ $\pi_{x,1}$ $\pi_{1,2}$ $\pi_{x,y}$
$\pi_{x1,y1}$ $\pi_{x2,y2}$ $\pi_{x3,y3}$ … $\pi_{xp,yp}$ $P_{\pi1,1}$ $P_{\pi2,1}$ … $P_{\pi x,1}$ $P_{\pi1,2}$ … $P_{\pi x,y}$
$\pi_{1,y}$
$\pi_{2,y}$
$\pi_{x,y}$
$\pi_{1,2}$
$\pi_{2,2}$
$\pi_{x,2}$
$\pi_{1,1}$
$\pi_{2,1}$
$\pi_{x,1}$

Figure 7 Genes in chromosomes

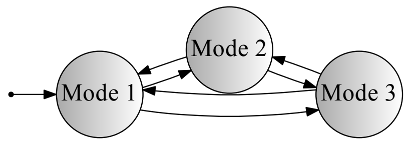

Figure 8 FSM describing modes in ECM

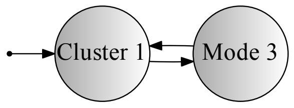

Figure 9 FSM describing clustered modes in ECM

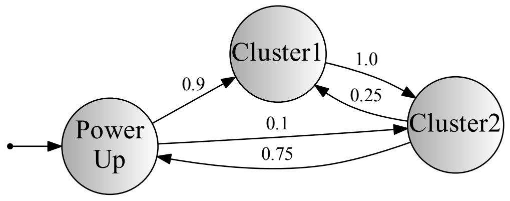

Figure 10 FSM describing clustered modes in DemoCar

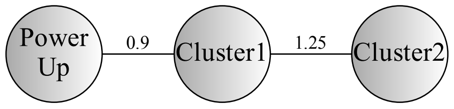

Figure 11 Spanning tree for clustered DemoCar

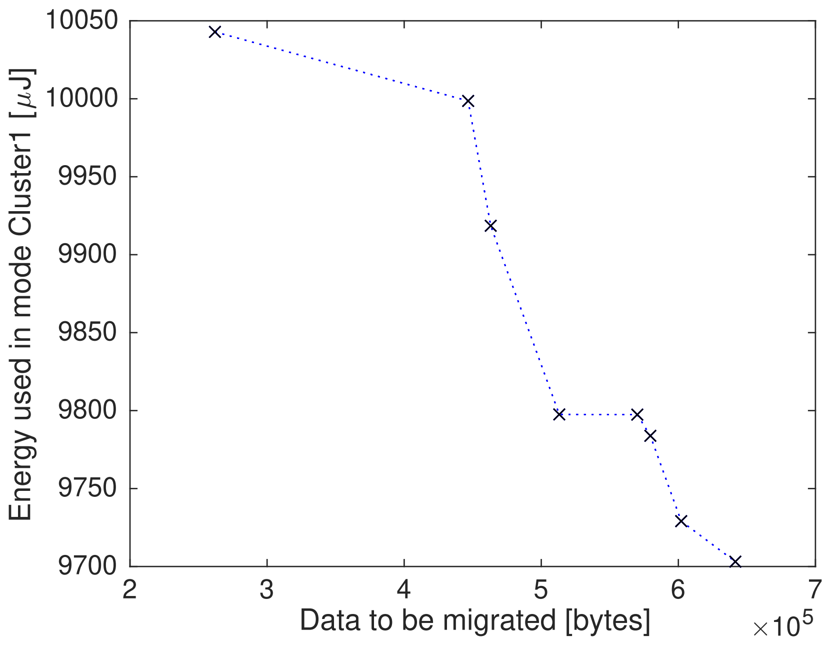

Figure 12 Pareto curve illustrating the trade-off between minimal amount of data to be migrated and minimal energy dissipated in the next mode