12. Power-aware Allocation of Fault-tolerant Multirate AUTOSAR Applications

Nesredin Mahmud, Guillermo Rodriguez-Navas, Hamid Faragardi, Saad Mubeen and Cristina Seceleanu. *In the 25th Asia-Pacific Software Engineering Conference (APSEC'18). IEEE.*

Abstract: Software-to-hardware allocation plays an important role in the development of resource-constrained automotive embedded systems that are required to meet timing, reliability and power requirements. This paper proposes an Integer Linear Programming optimization approach for the allocation of fault-tolerant embedded software applications that are developed using the AUTOSAR standard. The allocation takes into account the timing and reliability requirements of the multirate software applications and the heterogeneity of their execution platforms. The optimization objective is to minimize the total power consumption of the applications that are distributed over multiple computing units. The proposed approach is evaluated using a range of different software applications from the automotive domain, which are generated using the real-world automotive benchmark. The evaluation results indicate that our proposed allocation approach is effective while meeting the timing, reliability, and power requirements of the considered automotive software applications.

12.1 Introduction

The software-to-hardware allocation is a very important step during the development of automotive embedded systems. Basically, it allows the designer to explore system-level solutions that meet functional and extra-functional software requirements together with resource availability on the execution platform. Software allocation is a well-researched area in the domain of embedded systems, including in hardware/software co-design [40], platform-based system design [33] and the Y-chart design approach [22]. It is a type of bin-packing problem, and therefore finding an optimal solution, in the general case, is NP-hard [17]. The methods to solve such problems can be exact [32], which means solutions are guaranteed to be optimal, or heuristic, which deliver near-optimal solutions [13][6]. Exact methods such as Integer Linear Programming (ILP) [5] have been used widely in several resource optimization problems. In contrast to heuristic methods, ILP returns optimal solutions faster for relatively small problems [41]. However, many problems in real-time systems are nonlinear by nature [16], e.g., response time of cause-effect actions, system reliability, etc. To benefit from linear optimization techniques, non-linear functions are approximated using *Linearization* - a widely-used technique in the optimization of non-linear problems.

In case of fault-tolerance with replication [24], the search space to find the optimal allocation is increased due to the replicas. The search space becomes even larger if we assume that the real-time system executes over different sampling rates, known as *multirate* [37], in which case the feasible (timed) paths that pass through the different sampling points (or activation patterns) increase exponentially with the number of activation patterns increase. Furthermore, due to the different sampling rates that result in oversampling and undersampling effects, the timing analysis of signals propagation is complex [28]. Existing methods of software allocation lack exact results for the timing analysis of multirate systems.

In this paper, we propose an allocation scheme based on ILP for relatively small- and medium-sized fault-tolerant distributed applications, with the number of allocatable components not exceeding 15, operating-system tasks less than 100, and cause-effect chains in the range of 30 to 60. These parameters are deducted from the real-world automotive benchmark [25], and from previous experience in developing automotive systems and experiments. The applications are distributed over heterogeneous computing units that share a single network. The allocation aims for minimizing the total power consumption of the system while meeting timing and reliability requirements. Our proposed solution targets the automotive domain, in particular systems that conform to the AUTomotive Open System ARchitecture (AUTOSAR) standard. In comparison to related work [41][36][32], we consider a fault-tolerant and multirate system model. Furthermore, we follow a highly integrated approach in the allocation process, which includes response-time analysis (RTA), and utilization bound checking (UB), as well as bounding the level of fault tolerance via the imposed reliability requirement on the application. The main contributions of our work are: i) an ILP model for the allocation of a fault-tolerant multirate application on heterogeneous nodes with the objective of minimizing the total power consumption, and ii) an approach for reducing overhead of replications and cause-effect chains on the allocation of such applications.

Our approach is evaluated on synthetic automotive applications that are generated according to the real-world automotive benchmark proposed by Kramer et al. [25]. In the evaluation, we show the performance of our proposed approach in terms of allocation time and resource efficiency with respect to the size of applications. The tool and the synthetic applications used in this experiment are publicly available from BitBucket https://bitbucket.org/nasmdh/archsynapp/src/master/.

The rest of the paper is organized as follows. Section 12.2 provides a brief overview of AUTOSAR-based software development, emphasizing the role of software allocation. Section 12.3 describes the system model, and Section 12.4 describes the extra-functional models including the timing, reliability, and power consumption models. Section 12.5 presents the proposed allocation scheme, and in Section 12.6, we provide an evaluation of the proposed approach using the automotive benchmark. In Section 12.7, we compare to related work. Finally, we conclude the paper in Section 12.8, and outline the possible future work.

12.2 AUTOSAR

The AUTomotive Open System ARchitecture (AUTOSAR) partnership has defined the open standard AUTOSAR for automotive software architecture that enables manufacturers, suppliers, and tool developers to adopt shared development specifications, while allowing sufficient space for competitiveness. The specifications state standards and development methodologies on how to manage the growing complexity of Electronic/Electrical (E/E) systems, which take into account the flexibility of software development, portability of software applications, dependability, efficiency, etc., of automotive solutions. The conceptual separation of software applications from their infrastructure (or execution platform) is an important attribute of AUTOSAR and is realized through different functional abstractions [31].

12.2.1 Software Application

According to AUTOSAR, software applications are realized on different functional abstractions. The top-most functional abstraction, that is the Virtual Function Bus (VFB), defines a software application over a virtual communication bus using software components that communicate with each other via standard interfaces of various communication semantics. The behavior of a software component is realized by one or more atomic programs known as *Runnables*, which are entities that are scheduled for execution by the operating system and provide abstraction to operating system tasks, essentially enabling behavioral analysis of a software application at the VFB level. The Runtime Time Environment (RTE), which is the lower-level abstraction, realizes the communication between Runnables via RTE Application Programming Interface (API) calls that respond to events, e.g., timing. Furthermore, the RTE implementation provides software components with the access to basic software services, e.g., communication, micro-controller and ECU abstractions, etc., which are defined in the Basic Software (BSW) abstraction [31].

12.2.2 Timing and Reliability of Applications

The timing information of applications is a crucial input to the software allocation process. Among other extensions, the AUTOSAR Timing Extension specification [3] states the timing descriptions and constraints that can be imposed at the system-level via the *SystemTiming* element. The timing constraints realize the timing requirements on the observable occurrence of events of type *Timing Events*, e.g., Runnables execution time, and *Event Chains*, also referred to as *Cause-effect Chains* that denote the causal nature of the chain. In this work, we consider periodic events and cause-effect chains with different rates of execution (or activation patterns).

Although the importance of reliability is indicated in various AUTOSAR specifications via best practices, the lack of a comprehensive reliability design recommendations has opened an opportunity for flexible yet not standardized development approaches. In this paper, we consider application reliability as a user requirement and, in the allocation process, we aim at meeting the requirement via optimal placement and replication of software components.

12.3 System Model

The system model consists of three parts: a software application, an execution platform, and a software allocation scheme. An overview of the system model is illustrated in Figure 12.1. The software application is a user-defined software system, such as x-by-wire, electronic throttle control, flight control, etc., which is developed using software components [26][8]. The application is deployed on an execution platform, which is a network of heterogeneous nodes with possibly different processor frequencies, power consumption, and failure-rates. The allocation scheme, which defines a mapping relation from software components to computational nodes, guarantees that the extra-functional properties such as application reliability and timing requirements are met. Furthermore, it takes the optimization of power consumption as its objective in the allocation process.

In this work, we target AUTOSAR-based systems due to the increasing popularity of the specification standard in the automotive industry, and the challenges and opportunities that automotive industry is facing especially in resource optimization and dependability of automotive systems. Note that our

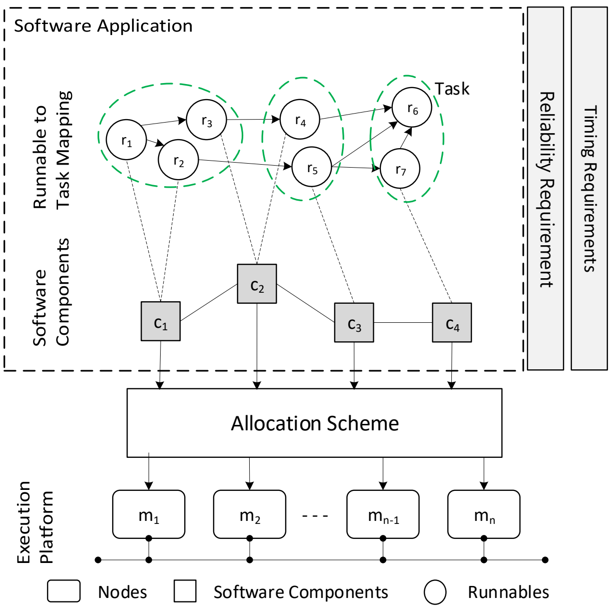


Figure 12.1: System Model.

approach can also be applied on different domains of distributed embedded systems with a slight change in the application modeling.

12.3.1 Software Application Model

In AUTOSAR-based systems, a software application is developed using AUTOSAR application software components C that consist of one or more runnables R_c [34]. In this notation, R_{c_i} refers to the set of runnables that are co-hosted in the component C_i.

Definition 1 (Software Application) *We define an AUTOSAR software application ζ as a Digraph (acyclic directed graph) $\langle V_r, E\rangle$ of runnable nodes V_r, where $\langle u, v\rangle \in E$ is a set of directed links from u to v, which denote the logical flow of the application, and $u, v \in V_r$. The runnable is a tuple $\langle e, p\rangle$, where $\bigcup_{i=1}^{N} e_i$ is a set of execution times, p is a periodic activation, and e_i refers to the execution time of the runnable r on the node m_i.*

The following assumptions are made in our proposed software allocation method:

- Allocatable software components are considered atomic, and therefore are allocated only on a single node, whereas, composite components need to be flattened first into their respective constituents of atomic components.
- Runnables are activated either periodically by clock events, or by predecessor runnables.

- Three cases of mapping runnables to tasks are considered: i) a runnable is mapped to a single task, ii) runnables that are collocated on the same software component are mapped to a single task, iii) runnables with the same activation periods and with triggering dependency are mapped to a single task. For more information, please see the AUTOSAR documentation [2].
- We assume that the computation nodes have the same types of interfaces. If this is not the case, software components can be constrained to nodes that the component supports.

12.3.2 Fault-tolerant Software Application Model

Redundancy is the most common way to increase the reliability of an application. It can be implemented according to different schemes, such as hot stand-by, cold stand-by, etc [10]. In this work the details of the redundancy scheme are abstracted away under the following assumptions: i) Hot stand-by redundancy technique is used for the replacement of failed components, which are identical and are allocated on different nodes, ii) software components need to be replicated if the application's reliability requirement is not met without replication, otherwise they are not replicated, iii) the time needed to detect and replace a faulty component is considered negligible and will not be taken into account in the response time analysis of tasks and delay calculation of cause-effect chains, iv) Because of its simplicity, the mechanism for detection and replacement of faulty components will be considered fault-free, and therefore will not be included in the reliability calculations.

We denote the k^{th} replica of a software component c as c^k, with $1 \leq k \leq K$; where K is the maximum number of replicas allowed for each application component.

12.3.3 Platform Model

The application is deployed on a network of heterogeneous computing nodes that are connected via a reliable communication network, the CAN bus. The computation node is specified as a 3-tuple $\langle hz, \lambda, p \rangle$, respectively, refer to the processor frequency, failure-rate and power consumption of a computation node. Due to the heterogeneity assumption of the processors, an application maybe be deployed on nodes with higher processor frequencies, and therefore fewer number of nodes in order to minimize the total power consumption of the system. However, due to the application reliability requirement, the application could be deployed differently, and with more resources. The CAN bus is considered reliable, for instance through redundancy. Therefore, its exclusion from the overall calculation of the system's reliability does not impact our proposed software allocation.

12.4 Modeling Extra-functional Properties

In this section, we discuss the timing, application reliability, and power consumption models used throughout the paper.

12.4.1 Power Consumption Model

Power consumption refers to the energy usage of electronic components in an integrated circuit, e.g., processor, memory, I/O devices, etc., per time unit. Depending on the nature of the integrated circuit and intended use, there exist low-level and high-level models of power and energy consumption estimation techniques. The low-level models, for instance in CMOS-based integrated circuits estimate power consumption via the power consumption of flip-flops and combinatorial gates [30][29], and they are frequently used in the design of power-efficient electronic circuits designs. The high-level models apply dynamic profiling of computer components, e.g., CPU, memory, I/O devices, etc., to estimate power consumption of the computer system, and they are primarily used in energy management techniques, e.g., in dynamic Voltage and Frequency Scaling (DVFS) [7].

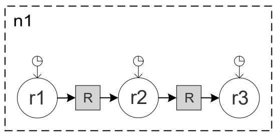
However, the previously mentioned power estimation techniques have limitations especially for applications in the early stages of system design. Their limitations are as follows: i) the lack of complete and accurate information of electrical specification of integrated circuits makes the use of low-level power estimation methods difficult, and ii) the dynamic profiling of high-level techniques requires runtime mechanisms, such as performance counter monitor, which is not applicable in our case. Instead, in this work, we employ a different approach that is based on processor load (or *Processor Utilization*) to estimate the average power consumption of a computational node. Specifically, we use the linear polynomial model proposed by Fan et al. [12], which is shown in (12.1). The mode states that the power consumption of a node is directly proportional to its load, and is inductively formulated from experimental results:

$$p(u) = P_{idle} + (P_{busy} - P_{idle}) * u, \tag{12.1}$$

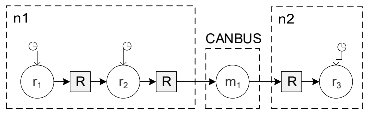
where u is the utilization (or load) of a computation node, p_{idle} and p_{busy}, respectively, refer to the power consumption of a node measured at minimum and maximum processor loads. Such measurements can be obtained by running performance benchmark suits, e.g., MiBench [19], AutoBench [11], etc.

12.4.2 Software Application Reliability

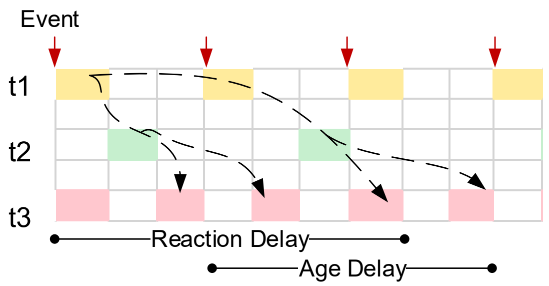
Application reliability R_a refers to the probability that a software application functions correctly by the time t, or within the time interval $[0,t]$ [18]. We assume that applications are free from design errors and, therefore, an application failure can be caused only by failures from the computational nodes in which the application is deployed. The failure rate of a node over



(a) Deployment on a Single Node.



(b) Deployment on Multiple Nodes.



(c) Timed Paths of Age and Reaction Delays.

Figure 12.2: Cause-effect Chain with Three Activation Patterns.

time is represented by $\lambda(t)$, and the reliability of a node is represented by the exponential density function over constant failure rate $\lambda e^{-\lambda t}$, where $\lambda = \lambda(t)$.

In a system without replication, the failure of any arbitrary node that hosts the software application renders the whole application faulty. Reliability calculation is then straightforward, using a series-parallel model: $R_a = \prod_{m \in M} r_m$. However, with the introduction of replication, to enable fault tolerance, the reliability calculation is not straightforward due to the replicas of the same software component allocated to different nodes that result in functional interdependencies between nodes. A software application functions *correctly* if each software component is executed at least by one non-faulty node, and will be *faulty* otherwise (i.e., if there are one or more software components that are allocated only to faulty nodes and thus these components cannot be executed correctly).

To calculate the reliability in such cases, we use state enumeration, which is one of the reliability-preserving methods that are used to compute the reliability of a system with dependent components (or subsystems) [27]. The state enumeration method allows the exploration of all possible states of a system in the probability space *PS*. Our goal is to differentiate between the states in which the application functions, denoted by *Functions(s)*, and the states in which the application fails, denoted by *Fails(s)*. The application reliability R_a is then calculated as follows:

$$R_a = \sum_{s \in PS | Functions(s)} p_s = 1 - \sum_{s \in PS | Fails(s)} p_s \tag{12.2}$$

To obtain p_s, that means the probability that the application is in state $s \in PS$, we define the Boolean variable $z_m \in \{0,1\}$ to indicate whether a node $m \in$

M is either faulty, $z_m = 0$, or not, $z_m = 1$. Then, the probability is calculated using (´12.3).

$$p_s = \prod_{m \in M} ((z_m * r_m) + (1 - z_m) * (1 - r_m)), \qquad (12.3)$$

where r_m and $1 - r_m$ are a computation node's reliability and probability of failure, respectively.

12.4.3 The Timing Model

The software application can be considered as a set of cause-effect chains, which are directed paths in the application graph, e.g., activation of a cruise control system by pressing a rotary-wheel on the dashboard, slowing down of a car by pressing the brake pedal, etc. Each cause-effect chain is annotated with an end-to-end timing requirement that specifies the maximum time between a stimulus and the corresponding response of a chain. A cause-effect chain can be hosted on a single node or multiple nodes as illustrated in Figure 12.2a and Figure 12.2b, respectively. Moreover, it can be activated by a single activation pattern, or multiple activation patterns (multirate).

The calculation of data-propagation delays in multirate software applications is not trivial due to the oversampling and undersampling effects, caused by the different activation patterns. Consequently, there are different delay semantics, which differ depending on the timed paths through which the data is propagated from the input to the output of the chains [28]. In this work, we focus on the *age* and *reaction* delays, which are the most widely used delay semantics in the automotive embedded systems. The two delays in a cause-effect chain that is distributed over two nodes are demonstrated in Figure 12.2c. The tasks t1 and t2 execute on one node, whereas task t3 executes on the second node. Note that t2 communicates with t3 via a network message, that is not shown in the figure for simplicity. The red inverted arrows in Figure 12.2c represent the arrival of events at the input of the chain, whereas the dashed-curve arrows represent the timed paths through which the data propagates from the input to the output of the chain. The age delay is the time elapsed between a stimulus and its corresponding latest non-overwritten response, i.e., between the 2^{nd} instance of t1 and the 5^{th} instance of t3. This delay is frequently used in the control systems applications where freshness of data is paramount. For example, the torque applied to turn the wheels must correspond to the position of the steering wheel and must be time bound. The reaction delay is the earliest time the system takes to respond to a stimulus that "just missed" the read access at the input of the chain. Assume that an event occurs just after the start of execution of the 1^{st} instance of t1. The data corresponding to this event is not read by the current instance of t1. In fact, the data will be read by the 2^{nd} instance of t1. The earliest effect of this event at the output of the chain will appear at the 4^{th} instance of t3, which represents the reaction delay. This delay is useful in body-electronics domain where first reaction to events is important, e.g., in the button-to-reaction applications. We refer the reader to [28] for the formal semantics of the two delays used in this paper.

12.5 Software Allocation Problem

In this section, we show our ILP model and the software-to-hardware allocation of a fault-tolerant application on heterogeneous nodes. Equation (12.4) defines the objective function for power consumption, with constraints on timing (12.5-12.6) and application reliability (12.7). The timing constraints consist of meeting the individual tasks deadlines *Deadline* as well as the end-to-end timing requirements of cause-effect chains *E2eReq* (12.6) in the distributed system. The reliability constraint ensures that a feasible solution meets the application reliability requirement *RelReq*.

$$\min_{x \in X} P(x) \tag{12.4}$$

$$\text{Subjected to:}$$

$$ResponseTime(x) \leq Deadline \tag{12.5}$$

$$Delay(x) \leq E2eReq \tag{12.6}$$

$$Reliability(x) \leq RelReq, \tag{12.7}$$

where x is a 3-dimensional binary matrix that represents a feasible solution, x_{ij}^k refers to the allocation of a software component c_i^k on node m_j, c^k refers to the k^{th} replica of c, and X is the search space of the function P.

In order to demonstrate our ILP optimization, we use a simple example throughout the section, which consist of software application and platform specifications: the software application is constructed from the set of components $C = \{c_1, c_2, c_3, c_4, c_5\}$, with maximum number of replicas $K = 2$. The application is deployed on nodes $M = \{m_1, m_2, m_3\}$. A detailed specification of the components and the nodes are shown in Table 12.1 and Table 12.3, respectively. A feasible solution x to the problem is shown in Figure 12.4.

M	$[P_{idle}, P_{max}, \lambda]$
m_1	[50.0, 140.0, 1.0E-3]
m_2	[10.0, 100.0, 1.0E-4]
m_3	[10.0, 140.0, 1 .0E-5]

Figure 12.3: Specification of Nodes.

$$\begin{array}{cc} x^1 & x^2 \\ \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \end{array}$$

Figure 12.4: A Feasible Solution x for K=2.

In the subsequent subsections, we explain the Integer Linear model, including the objective function and the constraints.

C	$R = [\text{ execution time} - (e_{m1}, e_{m2}, e_{m3}), period]$
c1	[(0.030, 0.060, 0.090), 1], [(0.041, 0.081, 0.122), 2] [(0.083, 0.167, 0.250), 5], [(0.310, 0.620, 0.930), 10]
c2	[(0.310, 0.620, 0.930), 10], [(0.310, 0.620, 0.930) 10] [(0.310, 0.620, 0.930), 10], [(0.310, 0.620, 0.930), 10]
c3	[(0.310, 0.620, 0.930), 10], [(0.291, 0.583, 0.874), 20] [(0.291, 0.583, 0.874), 20], [(0.291, 0.583, 0.874), 20]
c4	[(0.291, 0.583, 0.874), 20], [(0.291, 0.583, 0.874), 20] [(0.291, 0.583, 0.874), 20], [(0.093, 0.186, 0.279), 50]
c5	[(0.420, 0.841, 1.261), 100], [(0.420, 0.841, 1.261), 100] [(0.420, 0.841, 1.261), 100], [(0.420, 0.841, 1.261), 100]

Table 12.1: *Specification of Components.*

12.5.1 Power Consumption Optimization

The ILP model of the objective function is shown in (12.8-12.11). For a feasible solution x, the power consumption $P_{total}(x)$ is computed as the sum of the average power consumption of individual nodes $P_m(x)$. In order to compute the average power consumption of a node, first the node utilization is calculated using (12.10), which is the sum of the components' utilization (including replicas) that are allocated to that node. A component's utilization is obtained from the sum of sum of the utilization of the tasks that realize the component's functionality as shown in (12.11).

$$P_{total}(x) = \sum p_{m_j}(x) \tag{12.8}$$

$$p_{m_j}(x) = p(u_{m_j}(x)) \tag{12.9}$$

$$u_{m_j}(x) = \sum_k \sum_i u_{c_i} * x_{ij}^k \tag{12.10}$$

$$u_c = \sum_{\tau \in T_c} \frac{Exec(\tau_{m_j})}{Period(\tau)} \tag{12.11}$$

Table 12.2 illustrates the power consumption calculation of the software allocation example for the feasible solution (12.4).

In the ideal case, the minimum power consumption of the distributed system is achieved by centralizing the application on fewer nodes. However, due to the timing and reliability constraints, which require additional computing resources, the optimal solution could result in more used nodes.

M	C	$U_c(x)$	$P_m(x)$
m_1	$[c_2^1]$	[0.046, 0.017]	61.155W
m_2	$[c_1^1, c_2^2, c_3^1, c_4^1, c_5^1]$	[0.196, 0.248, 0.149, 0.091, 0.034]	114.648W
m_3	$[c_1^2, c_3^2, c_4^2, c_5^2]$	[0.224, 0.137, 0.050]	131.731W
		Total Power Consumption	307.534W

Table 12.2: *Average and Total Power Consumption of Nodes.*

12.5.2 Software Application Reliability

An optimal solution x must fulfill the application reliability requirement RelReq, which is usually in the range $[0.999, 0.999999]$ for safety-critical applications. The ILP formulation of the application reliability model, which is shown in (12.2), is shown in (12.12).

$$Reliability(x) = \sum_{s \in PS} [f(x,s)] * p_s, \tag{12.12}$$

where $[f(x,s)]$ is an *Iverson function* that returns 0 if the proposition that the application functions in state s is *true*. Otherwise it returns 1 if the proposition that application functions in state s is *false* (or the application fails in state s is *true*). The application functions only if all of its constituent software components functions and fails if at least of one of its components fails as formulated in (12.13), via the floor function. A software component functions if there exists a node m_j that hosts the component's replica $x_{ij}^k = 1$ and at the same time the node functions $s_j = 1$, which is formulated in (12.14) via the ceiling function. The floor and ceiling functions are piecewise linear functions, and are linearized by the ILP solver.

$$f(x,s) = \left\lfloor \frac{\sum_i f_{c_i}(x,s)}{N} \right\rfloor = \begin{cases} 1 & \text{if } application \text{ functions} \\ 0 & \text{if } application \text{ fails} \end{cases} \tag{12.13}$$

$$f_{c_i}(x,s) = \left\lceil \frac{\sum_k \sum_j x_{ij}^k * s_j}{K} \right\rceil = \begin{cases} 1 & \text{if } c_i \text{ functions} \\ 0 & \text{if } c_i \text{ fails} \end{cases} \tag{12.14}$$

Table 12.3 demonstrates the application reliability calculation for the feasible solution x (12.4).

In the case that the application reliability could be met with less replications, there is no need to keep unnecessary component replicas in the system. To this end, our optimization algorithm imposes soft constraints for $k > 1$, which implies that replicas allocated on the same node are reduced to a single replica, essentially discarding the extra replicas by design, since the

s	p_s	$[f_{c_i}(x), i = 1,2,3,4]$	$f_a(x)$
000	0.0000000000	[0, 0, 0, 0]	0
001	0.0000000099	[0, 0, 0, 1]	0
010	0.0000000099	[1, 1, 1, 1]	1
011	0.0000999800	[1, 1, 1, 1]	1
100	0.0000000099	[1, 1, 1, 0]	0
101	0.0000999800	[1, 1, 1, 0]	0
110	0.0000999800	[1, 1, 1, 1]	1
111	0.9997000299	[1, 1, 1, 1]	1

Table 12.3: *Application Reliability Calculation using State Enumeration Method, R(x) = 0.9998999998.*

reliability does not improve following additional replicas on the same node, assuming our fault model.

12.5.3 Timing Constraints

The timing constraints ensure that the response times of the tasks realizing the distributed application meet their deadlines. Furthermore, they ensure that the cause-effect chains satisfy their respective end-to-end timing requirements, for all possible failure-modes of the system. The constraints are formulated as logical constraints in the ILP problem, as explained in the rest of this subsection.

Tasks Deadline Constraints

The following pseudo-code illustrates how the ILP logical constraints of the tasks deadlines are prepared. It explores all possible sets of components combinations (or partitions) that can potentially be allocated to a node. Only the sets that are schedulable are asserted as constraints of the optimization problem, which is explained as follows: Line (1) identifies the power set of the components *Par*, followed by synthesis of tasks models of each partition. Line (2) checks the tasks models' schedulability and returns a matrix M^T that indicates schedulability, which is *true* if the task model is schedulable and *false* if not schedulable. Line (3) generates an ILP partition expression E for each node, then Line (4-6) asserts the expressions to hold in the optimization for the partitions that are schedulable.

In general, the number of potential logical constraints grow exponentially, which is in $2^{|C|} * |M|$. However, the effective logical constraints that are eventually asserted are much lower, for two main reasons: 1) a portion of the tasks models are not schedulable, therefore eliminated from the power set, due to CPU utilization exceeding the bound, hence not satisfying the response time of either task in the partition; ii) a task model can be a super

Algorithm 1 Generate Task Partitions Constraints.

```
Input: $C,M$
Ensure: Optimization Satisfies Tasks Deadlines, $D$
1: $Par \Leftarrow 2^C$
2: $M^T \Leftarrow isSched(Par,M)$
3: $E \Leftarrow MilpParExp(x)$
4: for all $m \in M$ do
5:     $assertOR(M^T_m, E_m, true)$
6: end for
```

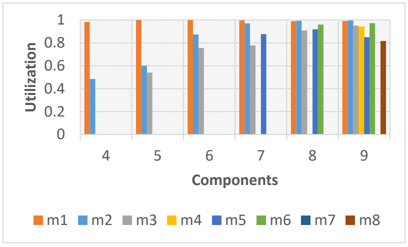
set of other tasks model. In this case, only the super model is checked, hence reducing pre-optimization time and logical constraints asserted in the solver.

Cause-effect Chains Constraints

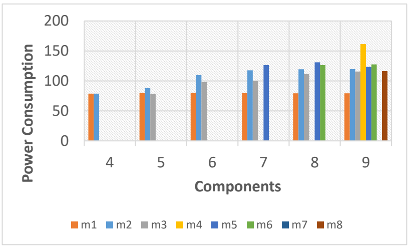
These constraints ensure that the cause-effect chains Γ meet their respective end-to-end requirements E2eReq. Similar to the previous constraints, the cause-effect chain constraints are logical assertions, which must be fulfilled by the optimal solution. The following pseudo-code illustrates how the ILP logical assertions are synthesized from the input models. The pseudo-code contains three main parts: i) the first part in Line (2) identifies the different deployment cases of the cause-effect chains over a set of nodes M, ii) the second part in Line (3-5), checks the schedulability of a deployable cause-effect chain ϕ against the reaction or age delays [28] and returns its schedulability matrix M^Γ, with values $true$ if schedulable and $false$ if not schedulable. For a schedulable ϕ, Line (5) constructs a conjunctive ILP expression that indicates the existence of at least one schedulable ϕ that satisfies the end-to-end requirement imposed on γ, and iii) the last part in Line (7) asserts the ILP logical OR expressions for each γ.

Algorithm 2 Generate Constraints on the Cause-effect Chains.

```
Require: $\Gamma,M$
Ensure: Optimization Satisfies End-to-end Requirements of Cause-effect
    Chains
1: for all $\gamma \in \Gamma$ do
2:     $\Phi \leftarrow Unique(C_r^{T_\Gamma}, M)$
3:     for all $\phi \in \Phi$ do
4:         $M^\Gamma \Leftarrow isSched(\phi,M)$
5:         $depExp \Leftarrow depExp \vee sched(M^\Gamma, true)$
6:     end for
7:     $assert(depExp)$
8: end for
```



(a) Utilization of Nodes.



(b) Power Consumption of Nodes.

Figure 12.5: Allocation of Applications on Heterogeneous Nodes.

12.6 Evaluation

In this section, we evaluate the proposed approach using synthesized automotive applications that conform to the automotive benchmark proposed by Kramel et al. [25]. The number of runnables, timing specification and activation patterns within the cause-effect chains are also selected according to the benchmark. In order to show the scalability of our approach and to assess the scope of its applicability in practice, in some cases, we use higher specifications standard than what is indicated in the benchmark, e.g., the maximum number of activation patterns is extended from three to four.

In the rest of the section, we describe the setup and method of evaluation, followed by discussion of the evaluation results.

12.6.1 Evaluation Setup

The evaluation setup consists of three hardware platforms with different computing capacities, i.e., processing speed and memory size as shown in Table 12.4. The evaluation on different platforms can be used as performance indicator and also to identify performance bottlenecks in the model. HP EliteBook and Lenovo 20378 are personal computers with core-i5 and core-i7 processors, respectively, whereas PowerEdge is a workstation with much higher processing and memory specification than the personal computers.

Hardware Model	Pro. Model	#Pro.	#Core	Cache	RAM
[1]HP EliteBook	[3]Core i5, 2.2GHz	1	2	3M	8G
Lenovo 20378	[4]Core i7, 2.6GHz	1	4	6M	16G
[2]PowerEdge	[5]Xeon(R), 2.4GHz	24	6	15M	256G

Table 12.4: *Summary of the Hardware Specifications.*
[1]*HP EliteBook 840 G2,* [2]*PowerEdge R730 Rack Server*
[3]*Intel® Core™ i7-4720HQ @ 2.60GHz,* [4]*Intel® Core™ i7-4720HQ @ 2.60GHz* [5]*Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz*

Software Application Specification

In order to evaluate the proposed approach on different ranges of applications, we automatically generate synthetic examples that comply with the automotive benchmark [25]. The examples denote simple to complex automotive functions such as reverse-parking assistance system and engine control system. For simplicity, we identify three classes of applications with different sizes and complexity, shown as tuple (c, r, t, g), respectively for the number of software components, runnables, tasks, and cause-effect chains. Table 12.5 shows the range of values used in the applications for evaluation.

Parameter	Spec.-I	spec.-II	spec.-III
components, c	≤ 10	≤ 15	> 15
runnables, r	≤ 50	≤ 100	> 100
tasks, t	≤ 30	≤ 60	> 60
cause-effect chains, g	≤ 30	≤ 40	> 60
activation-pattern		$[2, 3, 4]$	
share of activation-patterns		$[0.7, 0.2, 0.1]$	

Table 12.5: *Specification of the Applications for Evaluation.*

Platform Specification

The specification of nodes can be obtained from simulation, vendor product specification and experience. Figure 12.6 shows the range of values that are used in the nodes' specification. In the experiment, the values are randomly generated while respecting the benchmark.

Parameter	Range
nodes	$4 - 10$
power consumption (Watt), p	$10 - 200$
failure-rate (/Mhr), λ	$10^4 - 10^{-2}$
speed factor, hz	$0.0 - 1.0$

Table 12.6: *Range of Values for the Specification of Nodes.*

Method of Evaluation

We conduct three experiments that assess the proposed approach for scalability in terms of *allocation time*. The allocation time is defined as the time required to prepare and solve the allocation problem using the proposed ILP method. Furthermore, we show resource efficiency in terms of saving nodes (i.e., using smaller number of nodes). The experiments consist of: i)

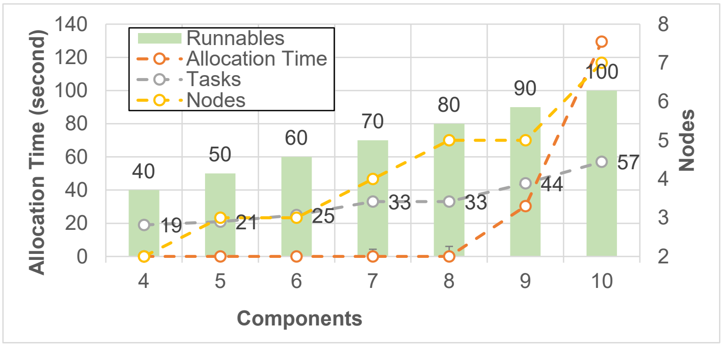


Figure 12.6: Effect of Varying the Application Size on the Allocation Time and Number of Utilized Nodes.

Node	m1	m2	m3	m4	m5	m6	m7	m8
P_{min}	10	40	30	20	30	40	20	10
P_{max}	80	120	120	170	140	130	140	110
λ	10^{-3}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-2}	10^{-6}	10^{-4}

Table 12.7: *Specification of Nodes.*

varying the size of applications in order to observe the effect of increasing components, runnables and tasks in the system, ii) varying the complexity of applications in order to observe the effect of cause-effect chains in the system, and iii) varying the replications. The experiments are discussed in detail in the next subsection.

A preliminary analysis of the evaluation indicates a successful termination of the allocation for Spec-I&II of the applications. Whereas for Spec-III, the allocation problem is intractable. In fact, it took days on the PowerEdge machine and many times it did not terminate successfully. Therefore, the experimental results that are shown in this paper are conducted on the Spec-I&II classes.

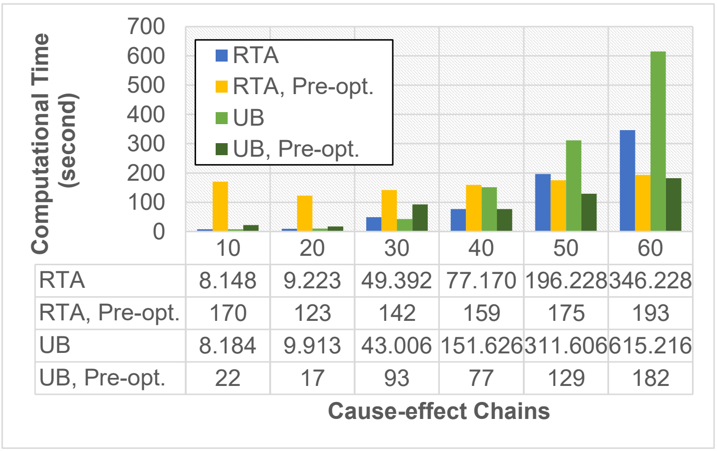
12.6.2 Varying the Size of Application

This refers to increasing the number of software components, as well as runnables, tasks, and cause-effect chains in the system. Figure 12.6 shows the effect of increasing the size of an application from $(c4, r40, t19, g30)$ to $(c10, r100, t57, g60)$ on the allocation time and the number of nodes utilized. The applications are allocated to a pool of 8 heterogeneous nodes sharing a single network and with specifications shown in Table 12.7. The specifications are generated randomly, with uniform distribution, within the scope of Table 12.6.

The CPLEX solver returns an optimal solution for the application $(c8, r80, t44, g30)$ within 6.06 sec. The allocation time increased sharply to 30.3 sec and 129.4 sec respectively for 9 and 10 components. Even if not indicated in this chart, the solver, the solver returns an optimal solution within 45 min for components reaching 15 on the PowerEdge machine and OutOfMemory error on the Lenovo and HP machines. Figure 12.5a and Figure 12.5b show the utilization and power consumption on each node for the different application sizes. The optimal allocation, in the general case, favor nodes with higher processor speed and lower power consumption specifications.

12.6.3 Varying the Number of Cause-effect Chains

In order to observe the effect of increasing the cause-effect chains on the allocation time, we vary the number of chains in the application from 10 to 60, which is consistent with the benchmark [25]. The share of activation patterns also increases proportionally with the ratio 1 : [0.7, 0.2, 0.1], respectively, for two, three, and four activation patterns. For instance, out of 10 cause-effect chains, there are 7 chains (with two activation patterns), 3 chains (with three activation patterns), and 1 chain (with four activation patterns). The experiment is conducted on two cases of schedulability analysis, namely response time analysis (RTA) and utilization bound (UB), and their result is shown in Figure 12.7 for increasing number of chains.



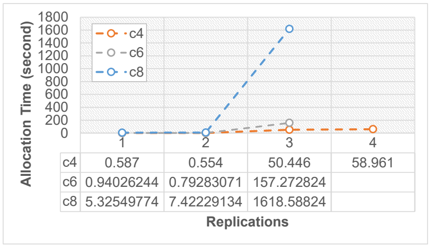
	10	20	30	40	50	60
RTA	8.148	9.223	49.392	77.170	196.228	346.228
RTA, Pre-opt.	170	123	142	159	175	193
UB	8.184	9.913	43.006	151.626	311.606	615.216
UB, Pre-opt.	22	17	93	77	129	182

Figure 12.7: Effect of Increasing Cause-effect Chains (under RTA and UB) on the Allocation Time.

The figures in the data table show an exponential growth of allocation time whenever the cause-effect chains are increased linearly for a specific application, in both cases of schedulability analysis. In the case of RTA, the overhead in the pre-optimization is higher than the overhead in the optimization for 50 or less chains. Whereas in the UB case, the computational time of pre-optimization is almost always less than the computational time of the optimization. The results are consistent with our expectation that the RTA computation, in the preparation of the timing assertions, is expensive, albeit provides schedulable tasks allocation based on the fixed-priority scheduling policy. In contrast, the UB computation time is relatively low; however, the search space gets larger due to more and more feasible tasks partitioning and chains fulfilling the timing constraints. As a result, the optimization time in the case of UB is usually higher as compared to the case of RTA. Therefore, for applications with chains not more than 40 and naive scheduling assumption using UB, the experiments favor the UB assumption. Whereas, the allocation with the RTA assumption should be selected for applications with exact scheduling requirements. Note, the scalability of this experiment should be seen in conjunction with the experiment discussed in the previous subsection.

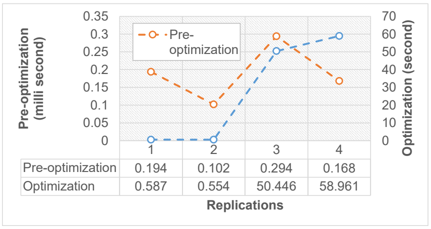
12.6.4 Varying Replications

In this experiment, we evaluate the allocation time of the applications with the increase in the number of replications. For the applications specification shown in Figure 12.8, we vary the replications from 1 to 4. The allocation in all applications took not more than 10 sec for replications 1 and 2. For Spec-I with replication 3 and 4, the allocation time went up close to 1 min. For Spec-III with replication 3, the allocation time went up rapidly to 30 min, and took extremely large time for replication 4 which is also the case for Application-II.



	1	2	3	4
c4	0.587	0.554	50.446	58.961
c6	0.94026244	0.79283071	157.272824	
c8	5.32549774	7.42229134	1618.58824	

(a) Effect on Allocation Time.



	1	2	3	4
Pre-optimization	0.194	0.102	0.294	0.168
Optimization	0.587	0.554	50.446	58.961

(b) Effect on Pre-optimization.

Figure 12.8: Effect of Varying the Component Replications on the Allocation Time.

Figure 12.8b shows the effect of shared constraints on the overhead of replications during the pre-optimization and optimization phases of the allocation. In the pre-optimization phase, the allocation time was stable for the increased size of applications. This is due to the fixed constraints regardless of the replications. In contrast, the allocation time increases during optimization as the constraints are applied for the various combination of task partitions, cause-effect chains, and reliability states generated as the result of replications.

12.7 Related Work

In a heterogeneous distributed system where computing nodes and communications links could have various failure rates, a reliability-aware allocation of tasks to nodes, and using links with the lowest failure rates can noticeably improve the system reliability [35][21][42][43]. Interleaving real-time constraints into the problem adds more complexity to reliability-aware task allocation in distributed systems [15]. As opposed to [41][32], we assume that software applications are multirate, which increase the difficulty of software allocation due the complexity of their timing analysis, and increased search space as the result of increasing timed paths of cause-effect chains. Furthermore, we assume a fault-tolerant system model.

Although improving reliability of the system using a reliability-aware task allocation does not impose extra hardware/software cost, in reliability-based design approach, redundancy (or replication) of software or hardware components is frequently applied to improve reliability. In such systems not only optimal allocation of software components (or replicas) should be taken into account but also the cardinality of the replicas should be limited for improved efficiency while meeting the desired reliability requirement. The integration of these two approaches (i.e., reliability-aware task allocation and application redundancy) is a promising technique to deal with high criticality of the system to fulfill the required reliability. For example, [1] proposes a heuristic algorithm to maximize reliability of a distributed system using task replication while at the same time minimizing the makespan of the given task set. Furthermore, in systems with replication, it uses the Minimal Cut Sets method, which is an approximate algorithm, to calculate reliability of a system. In contrast, we apply an exact method based on state enumeration, which is applicable to the problem size assumed in this work.

In our problem, power consumption is the other criterion of the optimization problem. Several research work exist on improving power consumption in real-time distributed systems. The research work [4] shows a survey of different methods on energy-aware scheduling of real-time systems, which categorizes the study into two major groups: i) Dynamic Voltage Scaling (DVS) [9][39], and ii) task consolidation to minimize the number of used computing and communication units [14], which is the approach followed in our work.

In the context of automotive systems, there are few works considering the reliability of a distributed system subject to real-time requirements of the automotive applications [20][23]. There are also other works discussing the allocation of software components onto nodes of a distributed real-time systems that consider other types of constraints other than reliability, for example, i) [38] which considers computation, communication and memory resources, and ii) [36] which proposes a genetic algorithm for a multi-criteria allocation of software components onto heterogeneous nodes including CPUs, GPUs, and FPGAs. Our approach also considers a hetrogeneous platform, i.e., nodes with different power consumption, failure-rate, and processor speed. In this work, we consider only the processor time; however, it can easily be extended to take into account different types of memory consumption that the software applications require.

12.8 Conclusions and Future Work

Software to hardware allocation plays an important role in the development of distributed and safety-critical embedded systems. Effective software allocation ensures that high-level software requirements such as timing and reliability are satisfied, and design and hardware constraints are met after allocation. In fault-tolerant multirate systems, finding an optimal allocation of a distributed software application is challenging, mainly due to the complexity of cause-effect chains' timing analysis, as well as the calculation of software application reliability. The timing analysis is complex due to oversampling and undersampling effects, caused by the different sampling rates, and the complexity of the reliability calculation is caused by the interdependency of the computation nodes due to replicas. Consequently, the formulation of the problem, to find an optimal solution, becomes non trivial.

In this work, we propose an ILP model of the software allocation problem for fault-tolerant multirate systems. The objective function of the optimization problem is minimization of power consumption with the aim of satisfying timing and reliability requirements, and meeting design and hardware constraints. The optimization problem involves linearization of the reliability model with piecewise functions, formulating the timing model using logical constraints, and limiting the number of replicas that can be used in the allocation. Furthermore, the allocation consider two cases of timing analysis: response time analysis and utilization bound.

Our approach is evaluated on synthetic automotive applications that are developed using the AUTOSAR standard, based on a real-world automotive benchmark. Although we consider automotive applications for the evaluation, the proposed approach is equally applicable to resource-constrained embedded systems, especially with timing, power and reliability requirements, in any other domain that are developed using the principles of model-based development and component-based software development. Our approach effectively applies to medium-sized automotive applications, but does not scale for complex applications. Considering similar system models, we plan to extend the current work with heuristic methods, e.g., genetic algorithms, simulated annealing, particle swarm optimization, etc., to handle large systems.

Acknowledgement

This work is supported by the Swedish Governmental Agency for Innovation Systems (Vinnova) through the VeriSpec project, and the Swedish Knowledge Foundation (KKS) through the projects HERO and DPAC.

12.9 References

[1] Ismail Assayad, Alain Girault, and Hamoudi Kalla. A Bi-criteria Scheduling Heuristic for Distributed Embedded Systems under Reliability and Real-time Constraints. In *Dependable Systems and Networks, 2004 International Conference on*, pages 347–356. IEEE, 2004.

[2] AUTOSAR. Specification of RTE Software. Technical report, AUTOSAR, 2017.

[3] AUTOSAR. Specification of Timing Extensions. Technical report, AUTOSAR, 2017.

[4] Mario Bambagini, Mauro Marinoni, Hakan Aydin, and Giorgio Buttazzo. Energy-aware Scheduling for Real-time Systems: A survey. *ACM Transactions on Embedded Computing Systems (TECS)*, 15(1):7, 2016.

[5] Hax Bradley. *Applied Mathematical Programming*. Addison-Wesley, 1977.

[6] Alessio Bucaioni, Lorenzo Addazi, Antonio Cicchetti, Federico Ciccozzi, Romina Eramo, Saad Mubeen, and Mikael Sjodin. MoVES: A Model-driven Methodology for Vehicular Embedded Systems. *IEEE Access*, 6:6424–6445, 2018.

[7] G. Contreras and M. Martonosi. Power Prediction for Intel XScale Processors using Performance Monitoring Unit Events. *ISLPED '05. Proceedings of the 2005 International Symposium on Low Power Electronics and Design, 2005.*, pages 221–226, 2005.

[8] Ivica Crnkovic, Magnus Larsson, and Inc Ebrary. Building Reliable Component-based Software Systems, 2002.

[9] Vinay Devadas and Hakan Aydin. On the Interplay of Voltage/Frequency Scaling and Device Power Management for Frame-based Real-time Embedded Applications. *IEEE Transactions on Computers*, 61(1):31–44, 2012.

[10] Elena Dubrova. *Fault-Tolerant Design*. Springer New York, New York, NY, 2013.

[11] EMBC. AutoBench™ 2.0 - Performance Suite for Multicore Automotive Processors, 2018.

[12] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power Provisioning for a Warehouse-sized Computer. *ACM SIGARCH Computer Architecture News*, 35(2):13, 2007.

[13] Hamid Reza faragardi, Björn Lisper, Kristian Sandström, and Thomas Nolte. A Resource Efficient Framework to Run Automotive Embedded Software on Multi-core ECUs. *Journal of Systems and Software*, 139:64–83, 2018.

[14] Hamid Reza Faragardi, Aboozar Rajabi, Reza Shojaee, and Thomas Nolte. Towards Energy-aware Resource Scheduling to Maximize Reliability in Cloud Computing Systems. In *High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on*, pages 1469–1479. IEEE, 2013.

[15] Hamid Reza Faragardi, Reza Shojaee, Mohammad Amin Keshtkar, and Hamid Tabani. Optimal Task Allocation for Maximizing Reliability in Distributed Real-time Systems. In *Computer and Information Science (ICIS), 2013 IEEE/ACIS 12th International Conference On*, pages 513–519. IEEE, 2013.

[16] Javier Fernandez, Cipriano Galindo, and Inmaculada García. *System Engineering and Automation An Interactive Educational Approach*. 2014.

[17] David Fernández-Baca. Allocating Modules to Processors in a Distributed System. *IEEE Transactions on Software Engineering*, 15(11):1427–1436, 11 1989.

[18] A.L. Goel. Software Reliability Models: Assumptions, Limitations, and Applicability. *IEEE Transactions on Software Engineering*, SE-11(12):1411–1423, 12 1985.

[19] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. MiBench: A Free, Commercially Representative Embedded Bbenchmark Suite. In *2001 IEEE International Workshop on Workload Characterization, WWC 2001*, pages 3–14, 2001.

[20] Shariful Islam, Robert Lindstrom, and Neeraj Suri. Dependability Driven Integration of Mixed Criticality SW Components. In *Object and Component-Oriented Real-Time Distributed Computing, 2006. ISORC 2006. Ninth IEEE International Symposium on*, pages 11–pp. IEEE, 2006.

[21] S Kartik and C Siva Ram Murthy. Task Allocation Algorithms for Maximizing Reliability of Distributed Computing Systems. *IEEE Transactions on computers*, 46(6):719–724, 1997.

[22] Bart Kienhuis, Ed F Deprettere, Pieter van der Wolf, and Kees Vissers. A Methodology to Design Programmable Embedded Systems. In *Embedded Processor Design Challenges: Systems, Architectures, Modeling, and Simulation — SAMOS*, pages 18–37. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[23] Junsung Kim, Gaurav Bhatia, Ragunathan Raj Rajkumar, and Markus Jochim. An Autosar-compliant Automotive Platform for Mmeeting Reliability and Timing Constraints. Technical report, SAE Technical Paper, 2011.

[24] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, W. Schwabl, C. Senft, and R. Zainlinger. Distributed Fault-tolerant Real-Time Systems: the Mars Approach. *IEEE Micro*, 9(1):25–40, 2 1989.

[25] Simon Kramer, Dirk Ziegenbein, and Arne Hamann. Real World Automotive Benchmarks for Free. In *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2015.

[26] Simone di Cola Kung-Kiu Lau. *What are Software Components?* World Scientific Publishing Company (June 29, 2017), 2017.

[27] Corinne Lucet and Jean-François Manouvrier. Exact Methods to Compute Network Reliability. In *Statistical and Probabilistic Models in Reliability*, pages 279–294. Birkhäuser Boston, Boston, MA, 1999.

[28] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Support for End-to-end Response-time and Delay Analysis in the Industrial Tool Suite: Issues, Experiences and A Case Study. *Computer Science and Information Systems*, 10(1):453–482, 2013.

[29] F N Najm. Power Estimation Techniques for Integrated Circuits. In *Computer-Aided Design, 1995. ICCAD-95. Digest of Technical Papers., 1995 IEEE/ACM International Conference on*, pages 492–499, 1995.

[30] F.N. Najm. A Survey of Power Estimation Techniques in VLSI Circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2(4):446–455, 1994.

[31] Nico Naumann. AUTOSAR Runtime Environment and Virtual Function Bus. *Hasso-Plattner-Institut, Tech. Rep.*

[32] Salah Eddine Saidi, Sylvain Cotard, Khaled Chaaban, and Kevin Marteil. An ILP Approach for Mapping AUTOSAR Runnables on Multi-core Architectures. In *Proceedings of the 2015 Workshop on Rapid Simulation and Performance Evaluation Methods and Tools - RAPIDO '15*, pages 1–8, New York, USA, 2015. ACM Press.

[33] Alberto Sangiovanni-Vincentelli, Luca Carloni, Fernando De Bernardinis, and Marco Sgroi. Benefits and Challenges for Platform-based Design. In *Proceedings of the 41st annual conference on Design automation - DAC '04*, page 409, New York, USA, 2004. ACM Press.

[34] Dietmar Schreiner and Karl M. Göschka. A Component Model for the AUTOSAR Virtual Function Bus. In *Proceedings - International Computer Software and Applications Conference*, volume 2, pages 635–641, 2007.

[35] Sol M Shatz, J-P Wang, and Masanori Goto. Task Allocation for Maximizing Reliability of Distributed Computer Systems. *IEEE Transactions on Computers*, 41(9):1156–1168, 1992.

[36] Ivan Švogor, Ivica Crnkovic, and Neven Vrcek. An Extended Model for Multi-Criteria Software Component Allocation on a Heterogeneous Embedded Platform. *Journal of computing and information technology*, 21(4):211–222, 2014.

[37] Luc Vinet and Alexei Zhedanov. A "Missing" Family of Classical Orthogonal Polynomials. *Computers as Components*, page 528, 11 2010.

[38] Shige Wang, Jeffrey R Merrick, and Kang G Shin. Component Allocation with Multiple Resource Constraints for Large Embedded Real-time Software Design. In *IEEE 10th Real-Time and Embedded Technology and Applications Symposium, 2004.*, pages 219–226. IEEE, 2004.

[39] Xi Wang, Imen Khemaissia, Mohamed Khalgui, ZhiWu Li, Olfa Mosbahi, and MengChu Zhou. Dynamic Low-power Reconfiguration of Real-time Systems with Periodic and Probabilistic Tasks. *IEEE Transactions on Automation Science and Engineering*, 12(1):258–271, 2015.

[40] W. Wolf. A Decade of Hardware/ Software Codesign. *Computer*, 36(4):38–43, 4 2003.

[41] Ernest Wozniak, Asma Mehiaoui, Chokri Mraidha, Sara Tucci-Piergiovanni, and Sébastien Gerard. An Optimization Approach for the Synthesis of AUTOSAR Architectures. In *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, 2013.

[42] Peng-Yeng Yin, Shiuh-Sheng Yu, Pei-Pei Wang, and Yi-Te Wang. Task Allocation for Maximizing Reliability of a Distributed System using Hybrid Particle Swarm Optimization. *Journal of Systems and Software*, 80(5):724–735, 2007.

[43] Longxin Zhang, Kenli Li, Yuming Xu, Jing Mei, Fan Zhang, and Keqin Li. Maximizing Reliability with Energy Conservation for Parallel Task Scheduling in a Heterogeneous Cluster. *Information Sciences*, 319:113–131, 2015.