An optimal approach to the task allocation problem on hierarchical architectures[*]

Alexander Metzner[*], Martin Fränzle[+], Christian Herde[+], Ingo Stierand[+]
[*]OFFIS, Oldenburg, Germany
[+]Dep. of Computer Science, Carl-von-Ossietzky University Oldenburg, Germany
email:{metzner,fraenzle,herde,stierand}@informatik.uni-oldenburg.de

Abstract

We present a SAT-based approach to the task and message allocation problem of distributed real-time systems with hierarchical architectures. In contrast to the heuristic approaches usually applied to this problem, our approach is guaranteed to find an optimal allocation for realistic task systems running on complex target architectures. Our method is based on the transformation of such scheduling problems into nonlinear integer optimization problems. The core of the numerical optimization procedure we use to discharge those problems is a solver for arbitrary Boolean combinations of integer constraints. Optimal solutions are obtained by imposing a binary search scheme on top of that solver. Experiments show the applicability of our approach to industrial-size task systems, which are mapped to heterogeneous hierarchical hardware architectures.

1 Introduction

Embedded systems typically have to satisfy strict timing requirements on using distributed architectures. In general, the task allocation problem is known to be NP-hard[1], and therefore assignments should be done automatically, guaranteeing the predictability with respect to real-time. Temporal predictability entails both the scheduling of tasks on ECUs (Embedded Control Units) as well as the transmission of messages over communication networks.

Predictable real-time behavior of task systems can be achieved through schedulability analysis, which has been investigated intensively in the past. [2] gives a survey on this topic. Accurate time prediction for the communication of messages on lifelike bus systems was shown akin to schedulability analysis of tasks in [3]. We will use these results providing a rich task model that enables the application of our method to real-world problems.

[*]This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS)

In the area of task allocation there are several approaches of mostly heuristic type for finding a mapping of tasks to ECUs and messages to networks. To simplify matters, several approaches restrict the architecture to a fixed specified interconnection of ECUs. For example in [4] the task allocation problem is investigated on architectures with a grid interconnection. Graph based heuristics are used to cluster tasks that are closely related in order to minimize communications. Unfortunately, neither exact timing predictions for the runtime of tasks, nor exact communication latencies are considered. Less stringent architectural restrictions are taken in [5], where a set of uniform ECUs is connected by a realistic bus system. Based on the results of [3], the allocation problem is attacked by a simulated annealing approach. Treatment of certain additional placement restrictions, like forbidden placements and memory consumption, is incorporated. However, this approach is not able to deal with hierarchical architectures (networks of ECUs with hierarchical topologies). The authors of [6] propose a combined branch & bound and list-scheduling approach to allocate tasks and messages of time-triggered and event-triggered task sets to ECUs with a time-triggered, TDMA-based bus system. Again, only very simple topologies are supported. In the area of synthesis for HW/SW co-design, some approaches include the task allocation problem, but usually use only very rough prediction of run-times and message latencies. As an example, the reader is referred to [7], where evolutionary algorithms perform an allocation of tasks to hardware or ECUs, thereby using a static schedule without preemption.

In contrast to such incomplete and suboptimal heuristic approaches we propose an optimal strategy to assign tasks and messages to ECUs and network busses in complex architectures. This is done by modeling timing and resource restrictions as a set of integer formulae. Adding a cost function reflecting the run-time overheads of a certain task allocation, we are able to determine the assignment of tasks with minimal cost. Optimal assignments are generated by a sequence of calls to an appropriate propositional SAT (Satisfiability) checker[8], after transformation of the integer formulae into propositional formulae over the booleans.

Optimal approaches for the task allocation problem were already proposed in [9] and [10]. In [9], mixed integer linear programming is used to synthesize a static schedule of tasks on an ECU in a HW/SW co-synthesis environment. Due to the restrictions of the co-synthesis domain, the task model is quite simple: all tasks have the same period and are non-preemptive and, therefore, do not suffice task models we aim. Furthermore communication overheads are not modeled exactly and are used only for simple direct links between ECUs. [10] presents an optimal branch and bound algorithm for the task allocation problem on a richer task model that allows non-uniform periods for tasks, but is restricted to non-preemptive scheduling, too. Again, the communication model is approximate.

Contrary to the last two approaches, we aim at static priority-driven preemptive schedules for tasks. Furthermore, our approach provides exact analysis of message delivery cost on different bus systems, like bus systems driven by time division multiple access (TDMA) or event-driven bus systems, like CAN. Additionally, we consider cost for messages arising while crossing gateway nodes.

The paper is organized in 7 sections. Section 2 describes the architectural and task model and derives safe predictions of computation times for tasks as well as for messages in a distributed architecture. Section 3 presents the transformation of the task allocation problem into a set of integer-arithmetic formulae and section 4 extends this encoding in order to deal with hierarchical architectures. Our optimization algorithm for integer formulae is shown in section 5. Evaluations in section 6 demonstrate the applicability of the approach and, finally, section 7 concludes the paper.

2 Basic model

In order to formalize the system behavior of a real-time system within the time domain, an abstract model has to be created which defines on one hand the system architecture and on the other hand a task model which describes the application running on that architecture, plus the timing constraints the application must fulfill. A system architecture consists of a number of ECUs and a number of communication media the ECUs are connected to. Thus the architecture is described by a tuple $\mathcal{A} = (P, K, \kappa)$, where P is the set of ECUs, and $K \subseteq 2^P$ is the set of communication media, where a communication medium $k = \{p_1, \ldots, p_j\} \in K$ connects all ECUs p_1 to p_j. Finally, κ defines the typical parameters of each medium (like frame sizes, access methods, transfer rates, etc.).
In order to perform the task allocation and the schedulability analysis some assumption about the underlying scheduling strategy must be made. In this paper tasks are scheduled by a preemptive, fixed priority algorithm[5].

A task may send messages at the end of each computation to one or more other tasks. The arrival of a messages on an ECU may activate the receiving task. The timing constrains exist for each task and each message. The task model is defined by a set $\mathcal{T}$ of tuples $\tau_i = (t_i, c_i, \gamma_i, \pi_i, \delta_i, d_i)$ describing the individual tasks. The elements are $t_i \in \mathbb{N}$ activation period or minimal inter-arrival time, $c_i : P \to \mathbb{N}$ worst case execution times (WCET), $\gamma_i \subseteq \mathcal{T} \times \mathbb{N} \times \mathbb{N}$ messages (including their target, their size and their deadline) the task is sending, and $\pi_i \subseteq P$ the ECUs the task is allowed to be allocated on. Tasks from δ_i are not allowed to be allocated together with τ_i (redundant tasks which are used in fault tolerant systems), and $d_i \in \mathbb{N}$ is the deadline of τ_i.
Task allocation is now defined by the following functions: $\Pi : \mathcal{T} \to P$ assigns each task in $\mathcal{T}$ to an ECU, $\Phi : \mathcal{T} \times \mathcal{T} \to \{0, 1\}$ defines a priority ordering of tasks, and $\Gamma : \left(\mathcal{T} \times \mathbb{N} \times \mathbb{N} \to 2^K\right)$ assigns each message to a set of communication media.
Given a certain allocation, scheduling analysis evaluates whether all tasks can meet their timing constraints. For each task its worst case response time is calculated. The same is done for message transmissions, but for each communication media the message uses. In the remainder of this section it is outlined how the analysis is defined.

The scheduling analysis for priority ordered execution in its simplest form can be expressed by the following well known fixed point equation (cf. [2]):

$$r_i^{n+1} = c_i + \sum_{j \in hp(i)} \left\lceil \frac{r_i^n}{t_j} \right\rceil c_j \tag{1}$$

where $hp(i)$ is the set of tasks running on the same ECU with higher priority, and r_i is the worst case response time of a task τ_i. The iteration either ends with some n if $r_i^{n+1} = r_i^n$ or if r_i^n exceeds the given deadline.

An appropriate way to model message transmission is to exploit the analogy between arbitration of a bus by messages and CPU arbitration of an ECU by tasks. In general, we distinguish two types of bus systems: In *priority driven* busses each message is assigned a priority. The bus protocol then grants the bus access to the ECU with the highest requested priority. The CAN bus is a familiar priority bus. In TDMA based bus systems the bandwidth is divided into time slots. Each ECU is assigned a unique slot where it may send messages exclusively. The token ring used in [5] and the TTP are examples for TDMA busses.

In order to model message transmission, each message is assigned a unique priority. Messages to be sent are stored in a priority-ordered queue. A more comprehensive outline of this mechanism is, for example, given in [3]. Using a priority driven bus, the time needed for the transmission of a message m_i now can be calculated by

$$r_{m_i}^{n+1} = \rho_{m_i} + I_{m_i} \text{ with } I_{m_i} = \sum_{m_j \in hp(m_i)} \left\lceil \frac{r_{m_i}^{n+1}}{t_{m_j}} \right\rceil \rho_{m_j} \tag{2}$$

where ρ_{m_i} is the time needed to transmit message m_i over the bus. The time t_{m_j} is inherited by the task sending the message and therefore equals to the period of m_j's sender.

TDMA based communication media introduces an additional blocking factor because a message may have to wait for the slot assigned to the ECU the message is sent from before it can be transmitted. For more details see [3].

$$r_{m_i}^{n+1} = \rho_{m_i} + I_{m_i} + \left\lceil \frac{r_{m_i}^n}{\Lambda} \right\rceil (\Lambda - \lambda(S(\Pi(\tau_i)))) \tag{3}$$

where Λ is the length of a TDMA round and $\lambda(S(\Pi(\tau_i)))$ is the length of the slot assigned to the ECU the sending task is assigned to. The additional delay to the message in each TDMA round represents the situation in which the own slot just has past.

Based on the calculated response times for tasks and messages, the feasibility of the schedule can be provided by comparing them with their deadlines.

Using the same encoding scheme, many more temporal properties like release jitter, blocking factors, etc., can be represented. While this is done in our actual model we refrain from expanding on this due to lack of space.

3 Transformation of the allocation problem

In order to find (and optimize) a valid assignment of tasks to ECUs, we generate an encoding of the effect of the mappings Π and Φ on the response times. The encoding is a set of arithmetic formulae over integers which are connected by conjunction. In the following, all variables of the formulae are denoted by using a `typewriter` font. All other identifiers are constants that can be calculated at transformation time.

Let $\mathtt{a}_i$ be the *allocation variable* for task τ_i. $\mathtt{a}_i$ ranges over $[p_0, \ldots, p_n]$ and determines, which ECU task τ_i is assigned to. Hence $\mathtt{a}_i$ represents $\Pi(\tau_i)$. The optimization procedure can assign arbitrary values to $\mathtt{a}_i$, but it has to consider the placement constraints and the set of redundant tasks. We formulate this by the following in-equations:

$$\left(\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall p \in P \setminus \pi_i} \mathtt{a}_i \neq p \right) \wedge \left(\bigwedge_{\forall \tau_i \in \mathcal{T} : \delta_i \neq \emptyset} \bigwedge_{\forall \tau_j \in \delta_i} \mathtt{a}_i \neq \mathtt{a}_j \right) \tag{4}$$

In order to guarantee real-time constraints given by deadlines, we have to transform the response time analysis from equation (1) into a set of in-equations. Firstly, depending on the allocation of τ_i, the WCET $\mathtt{wcet}_i$ must be calculated.

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall p \in P} (\mathtt{a}_i = p) \rightarrow \mathtt{wcet}_i = c_i(p) \tag{5}$$

Subsequently, the response time $\mathtt{r}_i$ according to equation (1) of a task τ_i can be modeled as

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \mathtt{r}_i = \mathtt{wcet}_i + \sum_{\forall \tau_j \in \mathcal{T}} \mathtt{pc}_i^j, \tag{6}$$

where $\mathtt{pc}_i^j$ describes the preemption costs induced by tasks with higher priority. For each task τ_j that is assigned to the ECU $\Pi(\tau_i)$ and that has a higher priority $\mathtt{p}_i^j$ than τ_i ($\mathtt{p}_i^j = 1$), the WCET multiplied by the number of preemptions (caused by this task) determines the interference cost. This can be expressed with the following set of equalities:

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall \tau_j \in \mathcal{T} \setminus \{\tau_i\}} \big(\mathtt{p}_i^j \wedge (\mathtt{a}_i = \mathtt{a}_j)\big) \rightarrow \mathtt{pc}_i^j = \mathtt{I}_i^j \cdot \mathtt{wcet}_j \tag{7}$$

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall \tau_j \in \mathcal{T} \setminus \{\tau_i\}} \neg\big(\mathtt{p}_i^j \wedge (\mathtt{a}_i = \mathtt{a}_j)\big) \rightarrow \mathtt{pc}_i^j = 0 \tag{8}$$

where $\mathtt{I}_i^j$ describes the number of preemptions of a task τ_i by a task τ_j. This number depends on the priority $\mathtt{p}_i^j$ of the pair of tasks. It is known, that the deadline monotonic approach is optimal if all deadlines are different. If two task's deadlines become equal, the optimization process should choose a unique priority assignment. This is done within the next two in-equations, where the priorities are assigned according to the deadline of the tasks. Equality of two deadlines enables an arbitrary, but consistent, assignment of `true` or `false` to $\mathtt{p}_i^j$.

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall \tau_j \in \mathcal{T} \setminus \{\tau_i\}} \mathtt{p}_i^j \geq 0 \wedge \mathtt{p}_i^j \leq 1 \wedge \mathtt{p}_i^j + \mathtt{p}_j^i = 1 \tag{9}$$

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall \tau_j \in \mathcal{T} \setminus \{\tau_i\}} \begin{aligned} &\big((\mathtt{d}_i > \mathtt{d}_j) \rightarrow \mathtt{p}_i^j = 1\big) \\ &\wedge \big((\mathtt{d}_i < \mathtt{d}_j) \rightarrow \mathtt{p}_i^j = 0\big) \end{aligned} \tag{10}$$

With the given priority and the given mapping $\Pi(\tau_i)$ we can now solve the recurrence equation (1), which is modeled in equation (6) using the number of preemptions $\mathtt{I}_i^j$ from each task in equation (7) and (8). $\mathtt{I}_i^j$ is a substitution of the ceiling function in (1) and we can derive the upper and lower bound of $\mathtt{I}_i^j$:

$$\left\lceil \frac{r_i}{t_j} \right\rceil =: \mathtt{I}_i^j \overset{def. \lceil\rceil}{\Longrightarrow} \frac{r_i}{t_j} \leq \mathtt{I}_i^j < \frac{r_i}{t_j} + 1$$

If we assign a value to $\mathtt{I}_i^j$, both bounds must hold. Hence we can transform both bounds into linear in-equations (a) and (b):

$$\frac{r_i}{t_j} \leq \mathtt{I}_i^j \Leftrightarrow r_i \leq \mathtt{I}_i^j \cdot t_j \tag{a}$$

$$\mathtt{I}_i^j < \frac{r_i}{t_j} + 1 \Leftrightarrow r_i > (\mathtt{I}_i^j - 1) \cdot t_j \tag{b}$$

Because $\mathtt{I}_i^j$ is an integer variable and taking the definition of the ceiling function into account, each assignment of $\mathtt{I}_i^j$, that satisfies conditions (a) and (b) in the above equations, equals the result of the term $\left\lceil \frac{r_i}{t_j} \right\rceil$. Furthermore, the structure of these conditions forces valid assignments to be a solution of the recurrence equation. Now these conditions can be translated into a set of inequalities for each task, where we have to ensure that tasks which are assigned to different

ECUs don't preempt each other. Avoidance of interference due to lower priorities is eliminated in equation (8).

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall \tau_j \in \mathcal{T} \setminus \{\tau_i\}} (\mathtt{a}_i = \mathtt{a}_j) \rightarrow ((\mathtt{I}_i^j \cdot t_j \geq \mathtt{r}_i) \wedge ((\mathtt{I}_i^j - 1) \cdot t_j < \mathtt{r}_i)) \tag{11}$$

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall \tau_j \in \mathcal{T} \setminus \{\tau_i\}} (\mathtt{a}_i \neq \mathtt{a}_j) \rightarrow \mathtt{I}_i^j = 0 \tag{12}$$

Finally, for each task we have to check whether the response time is less than or equal to the deadline of this task:

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \mathtt{r}_i \leq \mathtt{d}_i \tag{13}$$

As outlined in section 2, the calculation of response times of messages is quite similar to the determination of response times for tasks. Hence, we can abstain from expanding on this here. However, if we investigate the formulae for the response time of messages that are transmitted via TDMA-based communication media, there is the need to introduce a term for blocking caused by other slots (see equation 3). This is encoded in the following way:

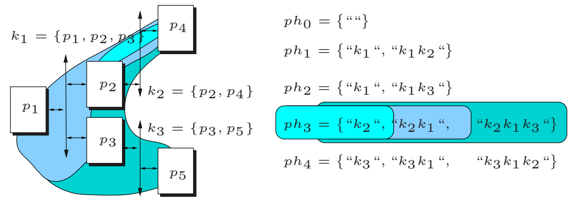
$$\bigwedge_{\forall \tau_i \in \mathcal{T} : \gamma_i \neq \emptyset} \bigwedge_{\forall m_j \in \gamma_i} \mathtt{block}_{m_j} = \mathtt{Imb}_{m_j} \cdot \left(\mathtt{tlen} - \mathtt{osl}_{m_j}\right),$$

where $\mathtt{tlen}$ is the length of the TDMA round, $\mathtt{Imb}_{m_j}$ contains the number of preemptions by other messages and $\mathtt{osl}_{m_j}$ is the length of the slot that $\Pi(\tau_i)$ is assigned to (if τ_i is the sender of message m_j). The interesting fact here is that the optimization procedure must assign values to all of these variables, thus we have to deal with *non-linear* (in)-equations in the optimization procedure (We will discuss the consequences in section 5).

4 Encoding for hierarchical architectures

Hierarchically organized architectures can be depicted as a set of graphs, possibly with cycles. Each node represents a communication medium and the arcs between nodes represent gateway ECUs that link media to each other. In the following we allow arbitrary networks, but only with one gateway between two media. Beside the generation of the mapping Π there is also a need for generating the mapping Γ during system construction. In order to determine the set of used media for each message m we introduce a boolean variable $\mathtt{K}_m^k$ that indicates the use of medium k by its truth value. As we will see below, this pure used/not used information is not sufficient: Firstly, we have to ensure that the used path is possible within the given topology. Secondly, successive message transmission on different media induces different jitter behavior, which we have to consider during schedulability analysis. Therefore, we have to know in which order the media are used.

For this purpose we introduce the so-called set PH of path closures. A path closure $ph \in PH$ is a set of ordered sets of media $''k_1 k_2 \ldots k_n''$, that contains all possible sub-paths of a path starting on a certain medium. Figure 1 demonstrates this on a simple example.



$k_1 = \{p_1, p_2, p_3\}$
$k_2 = \{p_2, p_4\}$
$k_3 = \{p_3, p_5\}$
p_1
p_2
p_3
p_4
p_5
$ph_0 = \{``"\}$
$ph_1 = \{``k_1", ``k_1 k_2"\}$
$ph_2 = \{``k_1", ``k_1 k_3"\}$
$ph_3 = \{``k_2", ``k_2 k_1", ``k_2 k_1 k_3"\}$
$ph_4 = \{``k_3", ``k_3 k_1", ``k_3 k_1 k_2"\}$

Figure 1. Path closures on a hierachical topology including the order of communication media in each possible path.

Each path closure represents a possible path on the topology and contains all sub-paths within these complete path. For example, in figure 1 the path closure that starts with medium k_2 and all its possible sub-paths within the hierarchical system topology are colored. While choosing one of these path closures for a message transmission during optimization, on the one hand we are able to check for the existence of the path chosen by the $\mathtt{K}$ variables, and on the other hand we have knowledge about the order of media within the chosen path. The implementation of Γ is realized by introducing path closure variables $\mathtt{Pf}_m$ over the set of path closures. After selecting one path closure for these variables, we have to check the compatibility of the path closure to the values of the $\mathtt{K}_m^k$ variables and the correctness of the allocation of sender and receiver task:

$$\bigwedge_{\forall ph_l \in PH} \bigwedge_{\forall \tau_i \in \mathcal{T} : \gamma_i \neq \emptyset} \bigwedge_{\forall m_j \in \gamma_i} (\mathtt{Pf}_{m_j} = ph_l) \rightarrow \bigvee_{\forall h \in ph_l} \left(\bigwedge_{k \in \bar{h}} \neg \mathtt{K}_{m_j}^k \bigwedge_{\forall 1 \leq l \leq |h| : k_l = pr_l(h)} \mathtt{K}_{m_j}^{k_l} \wedge v(h) \right) \tag{14}$$

where $\bar{h} = K \setminus h$ and $pr_j(h)$ returns the medium k of the j-th position in a path h. Equation 14 checks the correct usage and non-usage $\mathtt{K}_{m_j}^k$ of all communication media for each sub-path of the chosen path closure ph_l. Due to the construction of the sub-paths within a path closure the disjunction in equation 14 enables one and only one sub-path, whereas all other sub-paths are excluded. $v(h)$ checks whether the sender task is assigned to an ECU at the first medium and the target task is assigned to an ECU at the last medium of the path h, but not on gateway ECUs between media on the path:

$$v(h) := \begin{cases} \Pi(\tau_i) \in k_r \wedge \Pi(\tau_j) \in k_r & \text{if } h = \text{“}k_r\text{“} \\ \Pi(\tau_i) \in k_{r_1} \backslash (k_{r_1} \cap k_{r_2}) & \\ \wedge\ \Pi(\tau_j) \in k_{r_n} \backslash (k_{r_n} \cap k_{r_{n-1}}) & \text{else} \end{cases}$$

After determination of the used communication media, we know have to cater for deadlines of messages. The idea here is to generate local deadlines $\mathtt{d}^k_{m_t}$ for a message $m_t = (\tau_t, \cdot, \Delta_{m_t}) \in \gamma_i$ on each medium, s.t. the sum of all deadlines, and of the incurring cost ($\mathtt{serv}_{m_t}$) which raise from crossing gateway ECUs, equals the message's deadline:

$$\bigwedge_{\forall \tau_i \in \mathcal{T}: \gamma_i \neq \emptyset} \bigwedge_{\forall m_t \in \gamma_i} \left(\sum_{k \in K} \mathtt{d}^k_{m_t} + \mathtt{serv}_{m_t} \leq \Delta_{m_t} \right),$$

If a medium is not used, the deadline is forced to become 0:

$$\bigwedge_{\forall \tau_i \in \mathcal{T}: \gamma_i \neq \emptyset} \bigwedge_{\forall m_t \in \gamma_i} \bigwedge_{\forall k \in K} \left(\neg \mathtt{K}^k_{m_t} \rightarrow \mathtt{d}^k_{m_t} = 0 \right)$$

On each medium we now can perform schedulability analysis, but we have to consider the jitter of messages. The valuation of the jitter of a message on a certain medium k_i depends on the usage of all media k_j in front of k_i in the chosen path. Hence, the knowledge about the order of usage of media k_i within a path is essential for analysis purposes, and we provide this order by using path closures. The jitter can be calculated by the following formula:

$$\bigwedge_{\forall h \in PH} \bigwedge_{\forall \tau_i \in \mathcal{T}: \gamma_i \neq \emptyset} \bigwedge_{\forall m_j \in \gamma_i} (\mathtt{Pf}_{m_j} = h) \rightarrow$$
$$\bigwedge_{k \in \tilde{h}} \left(\mathtt{J}^k_{m_t} = J_{m_t} + \sum_{j=1}^{pos(k,\tilde{p})-1} \left(\mathtt{d}^{pr_j(\tilde{h})}_{m_t} - \beta_{pr_j(\tilde{h})}(m_t) \right) \right),$$

where $\mathtt{J}^k_{m_t}$ is the jitter of message m_t on medium k, J_{m_t} is the release jitter of m_t, $\tilde{h}$ is the longest path of a path closure h, $pos(k,h)$ is the position of medium k in the path h, and $\beta_k(m_t)$ is the best case transmission time of a message m_t on medium k. For each used medium k the jitter $\mathtt{J}^k_{m_t}$ is here defined by the sum of the difference of the worst case response time and the best case transmission time of all predecessor media of k, whereby the worst case response time is safely approximated by using the deadline $\mathtt{d}^{pr_j(\tilde{h})}_{m_t}$ of the message on the medium $pr_j(\tilde{h})$.

Finally, we are able to construct the response time analysis of messages as outlined in section 2 using the approach presented in section 3, but including jitter (cf. [2]). If $\mathtt{r}^k_m \leq \mathtt{d}^k_m$ holds for all used media k on a chosen path h, we have proven that sending the message m on path h is feasible.

Once we have done this modeling for tasks and messages, we can add an optimization function, for example minimizing utilization. For utilization optimization, for example, an in-equation is added which encodes that the difference to the average utilization is below some limit.

5 A SAT–based approach to integer optimization

The encoding sketched in the previous section reduces the problem of finding an optimal allocation of tasks to an optimization problem over the integers. The constraints of this optimization problem are given as a Boolean combination ϕ of linear and non-linear integer equations and in-equations, while optimality amounts to minimizing the value of some integer variable i occurring in ϕ subject to these constraints. This does in principle constitute a standard integer optimization problem, yet the formulae ϕ obtained from above encoding feature a peculiar structure which calls for extremely powerful optimization procedures:

- they are massively disjunctive, representing the plethora of choices entailed in the scheduling problem, and
- besides large linear (in)-equation systems over the integers, they do also contain a non-trivial number of non-linear integer constraints.

These formulae are thus hardly amenable to conventional numeric optimization procedures, which forced us to employ a reduction to propositional satisfiability solving (SAT) which is possible due to the bounded range of all integer variables entailed. We pursue a two-step approach to perform this reduction:

1. we encode arithmetic constraints over the integers by propositional formulae, enabling us to use a SAT checker to determine satisfying valuations for arbitrary Boolean combinations thereof, and
2. employ binary search for finding an optimal valuation.

Taking advantage of the recent advances in SAT [11, 12, 13] which have enhanced the size of tractable SAT problems by orders of magnitude within just a few years, such a reduction is able to tackle large instances of our scheduling problems.

5.1 Solving the arithmetic satisfiability problems

In order to discharge decision problems concerning satisfiability of arithmetic constraint systems ϕ over the integers, we employ the following reduction to purely Boolean satisfiability problems: First, helper-variables are introduced to obtain an equi-satisfiable problem that only contains “triplets”, i.e. is a Boolean combination of arithmetic equations and in-equations that comprise at most 3 variables, at most one binary operator, and exactly one relational operator. Thereafter, these arithmetic triplets are rewrote to propositional logic by using a 2's complement —and thus logarithmic size— representation for integer variables and a propositional axiomatization for the arithmetic operators on that representation.

Rewriting to triplet form. The rewriting to triplet form is basically similar to auxiliary variable introduction in the compilation of arithmetic expressions; in detail, it resembles the linear-time transformation of propositional formulae to CNF suggested by Tseitin in [14]. I.e., the overall formula ϕ is translated into the form $[\phi] \wedge T(\phi)$, where $[\phi]$ is a fresh propositional variable that represents the truth value of ϕ and

$$T(\phi \odot \psi) = ([\phi \odot \psi] \Leftrightarrow ([\psi] \odot [\psi])) \wedge \Theta(\phi, \psi) \qquad (15)$$

for arbitrary Boolean junctors $\odot$, with$[\phi]$ and $[\psi]$ again being fresh propositional variables and $\Theta(x,y) = T(x) \wedge T(y)$. Likewise,

$$T(e_1 \sim e_2) = ([e_1 \sim e_2] \Leftrightarrow [e_1] \sim [e_2]) \wedge \Theta(e_1, e_2) \qquad (16)$$

$$T(e_1 \otimes e_2) = ([e_1 \otimes e_2] = [e_1] \otimes [e_2]) \wedge \Theta(e_1, e_2) \qquad (17)$$

$$T(v) = v \quad \text{if } v \text{ is a variable} \qquad (18)$$

where $\sim$ is an arbitrary relational operator, $\otimes$ is an arbitrary arithmetic operator, $[e_1 \sim e_2]$ is a fresh propositional variable, $[e_i]$ and $[e_1 \otimes e_2]$ are fresh arithmetic variables, for which appropriate ranges are inferred from the ranges of the subexpressions. Such a rewriting yields an equisatisfiable formula where a satisfying assignment of the original formula can be extracted from a satisfying assignment of the triplet form by projection to the variables stemming from the original formula.

Encoding as propositional satisfiability problem. Once the triplet form is obtained, it remains to encode the arithmetic triplets, i.e. those obtained on transformations (16) and (17), by propositional logics. This amounts to replacing the arithmetic variables by an appropriate (wrt. 2's complement representation) number of propositional variables and the operators by an axiomatization of their counterparts on bit-vectors. For example, a triplet involving addition can be transformed to

$$\mathcal{P}(x+y=z) = \quad \bigwedge_{i=0}^{k} \quad \mathrm{fa}(z_i, c_{i+1}, x_i, y_i, c_i) \wedge \neg c_0 \wedge (z_{k+1} \Leftrightarrow c_{k+1}) \ ,$$

where k is the number of bits used for representing the values of x and y, c_0 to c_{k+1} are fresh propositional variables, and a fulladder *fa* is defined by

$$\mathrm{fa}(s, c_{\mathrm{out}}, x, y, c_{\mathrm{in}}) = \quad (s \Leftrightarrow (x \dot{\vee} y \dot{\vee} c_{\mathrm{in}})) \wedge (c_{\mathrm{out}} \Leftrightarrow (x + y + c_{\mathrm{in}} \geq 2)) \qquad (19)$$

It remains to encode the propositional formulae obtained from transformation $\mathcal{P}$ within the input language of a propositional SAT solver. To keep this encoding compact, we take advantage of Pseudo-Boolean formulae (PB) [15] rather than use an encoding by conjunctive normal form or by expressions built from binary Boolean operators, as often used in SAT solving. A PB formula is a conjunction of linear constraints over Boolean literals, i.e. similar to the constraint part of a 0-1 linear program, and allows to, e.g., encode the rightmost conjunct of (19) by $(2\overline{c_{\mathrm{out}}}+x+y+c_{\mathrm{in}} \geq 2) \wedge (2c_{\mathrm{out}} + \overline{x} + \overline{y} + \overline{c_{\mathrm{in}}} \geq 2)$, where $\overline{v}$ denotes the complement of v. We solve the resultant PB formulae by the Pseudo-Boolean constraint solver GOBLIN [8], which is the core of the HySAT tool [16]. The resultant formula is satisfiable iff the encoded integer-arithmetic constraint system ϕ is; furthermore, any satisfying assignment is a 2's complement representation of ϕ's triplet form s.t. a satisfying assignment of ϕ can be extracted by undoing the 2's complement encoding and projecting to ϕ's genuine variables.

5.2 Binary search for optimal solutions

Using above decision procedure, satisfiability of the constraint system ϕ can be decided and, if applicable, a satisfying assignment σ can be determined alongside with a corresponding valuation $\sigma(i)$ of the cost function i to be minimized. Hence, we can define a computable function SOLVE from integer constraint systems to $\mathbb{N} \cup \{-1\}$ s.t.

$$\text{SOLVE}(\phi) := \begin{cases} \sigma(i) & \text{for some } \sigma \text{ with } \sigma \models \phi \text{ if } \phi \text{ is satisfiable,} \\ -1 & \text{otherwise.} \end{cases}$$

Assume w.l.o.g. that the integer variable i we are going to minimize only takes values from $\mathbb{N}$. Then SOLVE yields an upper estimate of the cost of the optimal solution s.t. an optimal solution minimizing i can be obtained by applying a binary search scheme on the range of i as follows:

```
BIN_SEARCH($\phi$) :
    L := 0
    R := SOLVE($\phi$)
    while (L < R) do
        M := (L + R) div 2
        K := SOLVE($\phi \wedge i \geq$ L $\wedge i \leq$ M)
        if (K = $-1$) then L := M else R := K fi
    done
```

After termination of BIN_SEARCH, R either equals -1, indicating unsatisfiability of ϕ, or contains the optimal value of i. In the latter case, a corresponding task allocation can be obtained by solving $\phi \wedge i = o$, where o is the optimum of i, and extracting the placement and scheduling information from the satisfying assignment.

6 Experimental results

In order to verify the applicability and scaling of our approach, we ran several applications that are typical in their size for realistic task systems. The first example stems from

[5] and consists of 43 tasks, 12 task chains and some additional requirements (restricted placement, redundancies, memory consumption). The task set should be assigned to an architecture consisting of 8 ECUs, connected by a token-ring bus. [5] uses an optimization function, that minimizes the TDMA length of the token-ring (so called Token Rotation Time TRT). They apply a simulated annealing approach and report a result of $8.7msec$ for the smallest found TRT. Table 1 summarizes the results we yield with our SAT based approach and shows its complexity (number of boolean variables and literals).

Table 1. Experimental results based on the example given in [5].

Experiment	Result	Time	Var.	Lit.
[5]	$TRT = 8.55ms$	48 min	175k	995k
[5] + CAN	$U_{CAN} = 0.371$	361 min	298k	1627k

This example is a very tight one, thus there exist not many valid solutions. However, our optimal approach shows that simulated annealing in this case did not find the optimal solution, whereas SAT based methods are able to find the optimum ($8.55msec$) within an acceptable computation time interval. We also apply our technique to a CAN bus instead of the token-ring and are able to find an optimum. Here, the optimization goal was to minimize the bus load U_{CAN}. Due to the much more complex calculation of response times for messages in comparison to the simple token-ring protocol used in [5], the complexity of a model using CAN is much higher (see table 1). The complexity is given in forms of run-time, boolean variables and boolean literals of the propositional formulae extracted from the problem instance. The number of boolean variables reflects the state space the satisfiability checker has to evaluate, whereas from the number of boolean literals the number of constraints that restrict possible solutions can be derived. Due to the high dependability of most of the boolean variables from a small set of primary decision variables (e.g., the encoding of the $\mathtt{a}_i$ variables), the SAT checker is able to yield good runtimes on quite huge state spaces.

The growth of the complexity with respect to increasing task sets and increasing complexity of architectures is the objective of further evaluations, that we summarize in table 2 (a) and (b).

In table 2 the results are shown for an increasing complexity of architectures. The task set in this example consists of 30 tasks with several task chains and additional requirements. The architectural growth is instantiated by an increasing number of ECUs that are connected to a token-ring bus. For up to 20 ECUs the computation times are under one hour, which is very good at any rate for an optimal allocation approach. Beyond this limit, the computation times suffer from increasing complexity. However, even for industrial applications, the integration of architectures consisting of more than 20 ECUs from scratch is quite rare.

Table 2. Evaluations for the complexity related to the architectural size

ECUs	8	16	25	32	45	64
Time [h]	0:13	0:18	1:30	2:10	4:30	13:00
Var. (10^3)	100	133	148	158	178	206
Lit. (10^3)	602	814	911	979	1117	1304

Table 3. Evaluations for the complexity related to the size of task sets

Tasks	7	12	20	30	43
Time [h]	0:00:23	0:00:01	0:00:38	0:17	0:48
Var. (10^3)	5	14	34	88	174
Lit. (10^3)	22	74	191	492	995

Table 3 summarizes the effect of an increasing number of tasks. Here, we partitioned the example of [5] in smaller portions and applied the SAT based allocation approach. The computation times suffer from an almost exponential blow-up of the complexity. This is caused by the fact that the number of formulae directly depends on the number of tasks (in case of an architectural growth this is not the case). However, for up to 50 tasks the computation time is acceptable. Again, from an industrial point of view, task sets of this dimension are sufficient, because typically only parts of the complete system (so called functions or features) are integrated at a time (incremental integration).

For evaluation of hierarchical architectures we extend the architecture given in [5] by adding new communication media of type token-ring or CAN and introduce new ECUs that are used as gateway nodes. See figure 2 for an overview of the architectures under evaluation.

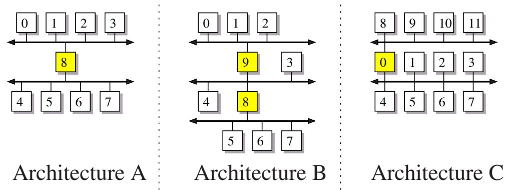


Figure 2. Hierarchical architectures

The results are reported in table 4, where the optimization goal was to minimize the sum of the TRT of all media. Gateway nodes in architecture A and B are not able to con-

Table 4. Results for the placement of the task set from [5] on architecture A to C.

Experiment	Result	Runtime
Arch A + [5]	$\sum TRT_i = 10.77ms$	490min
Arch B + [5]	$\sum TRT_i = 16.32ms$	740min
Arch C + [5]	$\sum TRT_i = 8.55ms$	790min

tain tasks of the task set. Hence, all tasks that are distributed over the border of sub-networks have to use two or three busses. Therefore, the result of the sum of TRT is greater than it was in the original system. On architecture C node 0 was used as gateway and the optimization yields the same placement of tasks than it yields for the original problem instance from [5]. We also exchange some of the busses with a CAN bus instead of a token-ring and this also leads to valid solutions. For example, exchanging the above media of architecture C by a CAN bus yields an optimal solution with $TRT = 8.55ms$ on the lower bus in about 180 minutes runtime. These experiments show, that our approach is not only able to produce optimal solutions for the task allocation problem on simple architectures, but we have also a support for heterogeneous hierarchical architectures.

7 Conclusion

We presented a SAT-based approach to the task allocation problem for distributed real-time systems of hierarchical architectures. In contrast to heuristic allocation approaches, which are known to behave well only if an application specific parametrisation of the algorithm can be found, our experiments show that we can achieve optimal assignments for industrial-sized applications from scratch, supporting accurate time predictions.

In particular, our approach is able to calculate task allocations for nearly arbitrary distributed, hierarchical networks, consisting of several different types of bus systems, where messages are allowed to be sent across more than one bus and induce additionally cost (e.g.) on gateway nodes.

A promising starting point for future work is to exploit the fact that our binary-search based optimization procedure entails the need to solve a sequence of SAT problems which only differ with respect to the constraints needed to confine the objective function to the respective search interval.

This enables the reuse of knowledge derived by the SAT solver's learning algorithm by simply copying the facts derived from a preceeding SAT instance to the subsequent instance, thus pruning the search space. First experiments show, that this technique is able to speedup the optimization procedure by a factor of 2 and more.

References

[1] J. Coffman. *Computer and Job-Shop Scheduling Theory*. John Wiley, 1976.

[2] N. Audsley, A. Burns, R. Davis, K. Tindell, and A. Wellings. Fixed priority pre-emptive scheduling: An historical perspective. *Real-Time Systems*, 8(2), 1995.

[3] K. Tindell. *Fixed Priority Scheduling of Hard Real-Time Systems*. PhD thesis, University of York, 1994.

[4] P. Altenbernd. *Timing Analysis, Scheduling, and Allocation of Periodic Hard Real-Time Tasks*. PhD thesis, Universität Paderborn, 1996.

[5] K. Tindell, A. Burns, and A. Wellings. Allocating hard real time tasks (an np-hard problem made easy). In *Real Time Systems Journal*, volume 4(2), 1992.

[6] P. Pop, P. Eles, Z. Peng, and V. Izosimov. Schedulability-driven partitioning and mapping for multi-cluster real-time systems. In *Proc. of the Euromicro Conference on Real-Time Systems*, 2004.

[7] T. Blickle, J. Teich, and L. Thiele. System-level synthesis using evolutionary algorithms. In *Proc. of the Design Automation for Embedded Systems*, 1998.

[8] M. Fränzle and C. Herde. Efficient SAT engines for concise logics: Accelerating proof search for zero-one linear constraint systems. In *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 2850 of *LNAI*. Springer, 2003.

[9] A. Bender. Design of an optimal loosely coupled heterogeneous multiprocessor system. In *Proc. of the European conference on Design and Test*. IEEE Computer Society, 1996.

[10] D. Peng, K. Shin, and T. Abdelzaher. Assignment and scheduling communicating periodic tasks in distributed real-time systems. *Software Engineering*, 23(12), 1997.

[11] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In *Proc. of the Design Automation Conference*, 2001.

[12] L. Zhang, C. F. Madigan, M. W. Moskewicz, and S. Malik. Efficient conflict driven learning in a Boolean satisfiability solver. In *Proc. of the International Conference on Computer-Aided Design*, 2001.

[13] E. Goldberg and Y. Novikov. Berkmin: A fast and robust sat solver. In *Design Automation and Test in Europe*, 2002.

[14] G. Tseitin. On the complexity of derivations in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logics*, 1968.

[15] P. Barth. A Davis-Putnam based enumeration algorithm for linear pseudo-boolean optimization. Technical Report MPI-I-95-2-003, MPI für Informatik, 1995.

[16] M. Fränzle and C. Herde. Efficient proof engines for bounded model checking of hybrid systems. In *International Workshop on Formal Methods for Industrial Critical Systems*, Electronic Notes in Theoretical Computer Science. Elsevier, 2004.