Optimization of Hierarchically Scheduled Heterogeneous Embedded Systems

Pop, Traian; Pop, Paul; Elecs, Petru; Peng, Zebo

Published in:
International Conference on Embedded and Real-Time Computing Systems and Applications

Link to article, DOI:
10.1109/RTCSA.2005.67

Publication date:
2005

Document Version
Publisher's PDF, also known as Version of record

Link back to DTU Orbit

Citation (APA):
Optimization of Hierarchically Scheduled Heterogeneous Embedded Systems

Traian Pop, Paul Pop, Petru Eles, Zebo Peng
Dept. of Computer and Information Science
Linköping University, SE-581 83, Linköping, Sweden
E-mail: {trapo, paupo, petel, zepbe}@ida.liu.se

Abstract
We present an approach to the analysis and optimization of heterogeneous distributed embedded systems for hard real-time applications. The systems are heterogeneous not only in terms of hardware components, but also in terms of communication protocols and scheduling policies. When several scheduling policies share a resource, they are organized in a hierarchy. In this paper, we address design problems that are characteristic to such hierarchically scheduled systems: assignment of scheduling policies to tasks, mapping of tasks to hardware components, and the scheduling of the activities. We present algorithms for solving these problems. Our heuristics are able to find schedulable implementations under limited resources, achieving an efficient utilization of the system.

1. Introduction
There has been a long debate in the real-time and embedded systems communities concerning the advantages and disadvantages of different scheduling approaches [1, 7, 9, 21]. Static cyclic scheduling (SCS) has the advantage of predictability and testability [9]. However, such static approaches lack the flexibility offered by event-driven approaches such as fixed priority scheduling (FPS) and earliest deadline first (EDF). EDF is optimal on single processor systems, and in general leads to a high, and thus efficient, resource utilization [7].

In this paper, we address design problems that are characteristic to such hierarchically scheduled systems: assignment of scheduling policies to tasks, mapping of tasks to hardware components, and the scheduling of the activities. We present algorithms for solving these problems. Our heuristics are able to find schedulable implementations under limited resources, achieving an efficient utilization of the system.

2. System Architecture
We consider architectures consisting of nodes connected by a unique broadcast communication channel (see Figure 1.a).

For the systems we are studying, we have designed a software architecture which runs on the CPU of each node. The main component of the software architecture is a real-time kernel. The real-time kernel contains three schedulers, for SCS, FPS, and EDF, organized hierarchically (Figure 1.c).

• First, we have extended our previous analysis approach [16] to handle hierarchically scheduled systems. We have proposed a holistic scheduling algorithm that builds the SCS tables and determines the FPS priorities, and provides a global analysis of the system.
• Once we can evaluate the schedulability of an implementation, we have developed an optimization approach to the assignment of scheduling policies to tasks and the mapping of tasks to the hardware nodes of the architecture.

This paper is organized in several sections. The next two sections present the hardware and software architectures considered, and the abstract model of the communication protocols. Section 4 presents the design optimization problem addressed, while Section 5 proposes optimization and scheduling algorithms for solving these problems. The optimization techniques are evaluated in Section 6, where we also discuss a real-life example. The last section presents our conclusions.
EDF priority levels within a task set on a processor. Higher priority EDF tasks can interrupt lower priority FFS tasks (as is the case with τ2 and τ3 which preempt τ4) and EDF tasks. Lower priority EDF tasks will be interrupted by both higher priority FFS and EDF tasks, and SCS tasks.

We model the bus access scheme using the Universal Communication Model [5]. The bus access is organized as consecutive cycles, each with the duration Tbus. We consider that the communication cycle is partitioned into static (ST) and dynamic (DYN) phases (Figure 1.b).

- ST phases consist of time slots, and during a slot only the node associated to that particular slot is allowed to transmit SCS messages. The transmission times of SCS messages are stored in a schedule table.
- During a DYN phase, all nodes are allowed to send messages and the conflicts between nodes trying to send simultaneously are solved by an arbitration mechanism which allows the transmission of the message with the highest priority. Hence, the ET messages are organized in a prioritized ready queue. The integration of EDF messages within such a priority-based arbitration mechanism has been detailed in [11].

TT activities are triggered based on a local clock available in each processing node. The synchronization of local clocks throughout the system is provided by the communication protocol.

3. Application Model
We model an application A as a set of directed, acyclic graphs \( G(V, E) \) ∈ A. A node \( τ_i \in V \) represents the j-th task in \( \mathcal{G} \). An edge \( e_{ij} \in \mathcal{E}_i \) from \( \tau_i \) to \( \tau_j \) indicates that the output of \( \tau_i \) is the input of \( \tau_j \). A task becomes ready after all its inputs have arrived and it issues its outputs when it terminates. The communication time between tasks mapped on the same processor is considered to be part of the task worst-case execution time and is not modeled explicitly. Communication between tasks mapped to different processors is performed by message passing over the bus. Such message passing is modeled as a communication task inserted on the arc connecting the sender and the receiver task.

Let \( \mathcal{P} \) be the set of tasks in A. The scheduling policy to be applied to each task is given by a function \( S : \mathcal{P} \rightarrow \{ \text{SCS, FPS, EDF} \} \). The mapping of a task \( \tau_i \in \mathcal{P} \) is given by a function \( M: \mathcal{P} \rightarrow \mathcal{N} \) where \( \mathcal{N} \) is the set of nodes in the architecture. For a task \( \tau_i \in \mathcal{P} \), \( M(\tau_i) \) is the node to which \( \tau_i \) is assigned for execution. Each task \( \tau_i \) can potentially be mapped on several nodes. Let \( \mathcal{N}_i \subseteq \mathcal{N} \) be the set of nodes to which \( \tau_i \) can potentially be mapped. We consider that for each \( \tau_i \in \mathcal{N}_i \), we know the worst-case execution time \( C_{\tau_i} \) on \( \mathcal{N}_i \). We also consider that the size of the messages is given (which can be directly converted into communication time on the particular bus). Tasks and messages activated based on events also have a priority, \( \text{Priority}_i \).

It is assumed that all tasks and messages in a task graph \( \mathcal{G}_i \) have the same period \( T_{\mathcal{G}_i} = 1 \) which is the period of the task graph. We consider that a deadline \( D_{\mathcal{G}_i} \) is given for each task graph \( \mathcal{G}_i \). In addition, tasks can have associated individual release times and deadlines. If communicating tasks are of different periods, they are combined into a merged graph capturing all tasks activations for the hyper-period (LCM of all periods).

4. Design Optimization Problems
Considering the type of applications and systems described in the previous section, several design optimization problems can be addressed. In this paper, we address problems which are characteristic to hierarchically scheduled distributed applications. In particular, we are interested in the following issues:

- assigning scheduling policies to tasks;
- mapping of tasks to the nodes of the architecture;
- optimization of the access to the communication infrastructure;
- scheduling of tasks and messages.

The goal is to produce an implementation which meets all the timing constraints of the application.

In this paper, by scheduling policy assignment (SPA) we denote the decision whether a certain task should be scheduled with SCS, FPS or EDF. Mapping a task means assigning it to a particular hardware node.

4.1 Scheduling Policy Assignment

Very often, the SPA and mapping decisions are taken based on the experience and preferences of the designer considering aspects like the functionality implemented by the task, the hardness of the constraints, sensitivity to jitter, etc. Moreover, due to legacy constraints, the mapping and scheduling policy of certain processes might be fixed. Thus, we denote with \( \mathcal{P}_{\text{SCS}} \subseteq \mathcal{P} \) the subset of tasks for which the designer has assigned SCS, \( \mathcal{P}_{\text{FPS}} \subseteq \mathcal{P} \) contains tasks to which FPS is assigned, while \( \mathcal{P}_{\text{EDF}} \subseteq \mathcal{P} \) contains those tasks for which the designer has decided to use the EDF scheduling policy. There are tasks, however, which do not exhibit certain features or requirements which obviously lead to their scheduling as SCS, FPS or EDF activities. The subset \( \mathcal{P}^* = \mathcal{P}_{\text{SCS}} \cup \mathcal{P}_{\text{FPS}} \cup \mathcal{P}_{\text{EDF}} \) of tasks could be assigned any scheduling policy. Decisions concerning the SPA to this set of activities can lead to various trade-offs concerning, for example, the schedulability properties of the system, the size of the schedule tables, the utilization of resources, etc.

Figure 2 shows an optimization example for the assignment of FPS and EDF policies. The application is composed of four tasks running on two nodes. Tasks \( \tau_1 \), \( \tau_2 \) and \( \tau_3 \) are mapped on node \( N_1 \), and have the same priority “1”, while task \( \tau_4 \) runs on \( N_2 \). Task \( \tau_4 \) is data dependent of task \( \tau_1 \). All tasks in the system have the same worst-case execution times (20 ms), deadlines (60 ms) and periods (80 ms). Tasks \( \tau_2 \) and \( \tau_3 \) are scheduled with EDF, \( \tau_4 \) with FPS, and we have to decide the scheduling policy for \( \tau_1 \) between EDF and FPS.

If \( \tau_1 \) is scheduled according to EDF, thus sharing the same priority level “1” with the tasks on node \( N_1 \), then \( \tau_1 \) misses its deadline (Figure 2.a). Note that in the time line for node \( N_1 \) in Figure 2 we depict several worst-case scenarios: each EDF task on node \( N_1 \) is depicted considering the worst-case interference from the other EDF tasks on \( N_1 \). However, the situation changes if on node \( N_1 \) we use FPS, i.e., changing the priority levels of \( \tau_2 \) and \( \tau_3 \) from “1” to “2”. Figure 2.b shows the response times when task \( \tau_1 \) has the highest priority on \( N_1 \) (\( \tau_1 \) retains priority “1”) and the other tasks are running under EDF at a lower priority level (\( \tau_2 \) and \( \tau_3 \) share lower priority “2”). Because in this situation there is no interference from tasks \( \tau_2 \) and \( \tau_3 \), the worst-case response time for task \( \tau_1 \) decreases considerably, allowing task \( \tau_1 \) to finish before its deadline, so that the system becomes schedulable.

4.2 Mapping and Bus Access Optimization

The designer might have already decided the mapping for a part of the tasks. For example, certain tasks, due to constraints such as having to be close to sensors/actuators, have to be physically located in a particular hardware unit. They represent the set \( \mathcal{P}_{\text{OK}} \subseteq \mathcal{P} \) of already mapped tasks. Consequently, we denote with \( \mathcal{P}^0 = \mathcal{P} \setminus \mathcal{P}_{\text{OK}} \) the tasks for which the mapping has not yet been decided.

The configuration of the bus access cycle has a strong impact on the global performance of the system. The parameters of this cycle have to be optimized such that they fit the particular application and the timing requirements. Parameters to be optimized are the number of static (ST) and dynamic (DYN) phases during a communication cycle, as well as the length and order of these phases. Considering the static phases, parameters to be fixed are the order, number, and length of slots assigned to the different nodes. Let us denote such a bus configuration with \( \mathcal{B} \).

4.3 Exact Problem Formulation

As an input we have an application \( A \) given as a set of graph tasks (Section 3) and a system architecture consisting of a set \( \mathcal{N} \) of nodes (Section 2). As introduced previously, \( \mathcal{P}_{\text{SCS}}, \mathcal{P}_{\text{FPS}} \) and \( \mathcal{P}_{\text{EDF}} \) are the sets of tasks for which the designer has already assigned SCS, FPS or EDF scheduling policy, respectively. Also, \( \mathcal{P}^0 \) is the set of already mapped tasks.

As part of our problem, we are interested to:

- find a SPA \( S \) for tasks in \( \mathcal{P}^0 \) in \( \mathcal{P} = \mathcal{P}_{\text{SCS}} \cup \mathcal{P}_{\text{FPS}} \cup \mathcal{P}_{\text{EDF}} \);
- decide a mapping for tasks in \( \mathcal{P}^0 \) in \( \mathcal{P} = \mathcal{P}_{\text{OK}} \cup \mathcal{P}^0 \);
- determine a bus configuration \( \mathcal{B} \);
- find a schedule table for the SCS tasks and priorities of FPS and EDF tasks such that imposed deadlines are guaranteed to be satisfied.

In this paper, we will consider the assignment of scheduling policies at the same time with the mapping of tasks to processors. Moreover, to simplify the presentation we will not discuss the optimization of the communication channel. Such an optimization can be performed with the techniques we have proposed in [16].

5. Design Optimization Strategy

The design problem formulated in the previous section is NP-complete (the scheduling subproblem, in a simpler context, is already NP-complete [20]). Therefore, our strategy, outlined in Figure 3, is to divide the problem into several, more manageable, subproblems. Our OptimizationStrategy has three steps:

1. In the first step (lines 1–3) we decide on an initial bus access configuration \( \mathcal{B}_0 \) (function InitialBusAccess), and an initial policy assignment \( S_0 \)
and mapping $\delta^k$ (function InitialMSPA). The initial bus access configuration (line 1) is determined, for the ST slots, by assigning in order nodes to the slots ($S_i = N$) and fixing the slot length to the minimal allowed value, which is equal to the length of the largest message in the application. Then, we add at the end of the TT slots an equal length single ET phase. The initial scheduling policy assignment and mapping algorithm (line 2 in Figure 3) maps tasks so that the amount of communication is minimized. The initial scheduling policy of tasks in $\mathcal{D}^s$ is set to FPS. Once an initial mapping, scheduling policy assignment and bus configuration are obtained, the application is scheduled using the HolisticScheduling algorithm (line 3) outlined in Section 5.1.

2. If the application is schedulable the optimization strategy stops. Otherwise, it continues with the second step by using an iterative improvement mapping and policy assignment heuristic, MSPAHuristic (line 4), presented in Section 5.2. to improve the partitioning and mapping obtained in the first step.

3. If the application is still unschedulable, we use, in the third step, the algorithm BusAccessOptimization presented in [16], which optimizes the access to the communication infrastructure (line 6). If the application is still unschedulable, we conclude that no satisfactory implementation could be found with the available amount of resources.

5.1 Holistic Scheduling

Once a partitioning and a mapping is decided, and a communication configuration is fixed, the tasks and messages have to be scheduled. For the SCS tasks this means building the schedule tables, while for the FPS tasks and EDF task groups, the priorities have to be determined and their schedulability has to be analyzed.

The basic idea is that SCS tasks are schedulable if it is possible to build a schedule table such that the timing requirements are satisfied. For FPS and EDF tasks, the answer whether or not they are schedulable is given by a schedulability analysis. In this paper, we use a response time analysis, where the schedulability test consists of the comparison between the worst-case response time $R_{ij}$ of a task $t_{ij}$ and its deadline $D_{ij}$. In order to derive the process of finding a schedulable system, which is presented in the next sections, it is not sufficient to test if the task set is schedulable or not, but we need a metric that captures the "degree of schedulability" of the application. For this purpose we use a cost function similar to the one described in [16].

The problem of finding a schedulable solution has to consider two aspects:

1. When performing the schedulability analysis for the FPS and EDF tasks and messages, one has to take into consideration the interference from the statically scheduled activities in the system.

2. Among the possible correct schedules for SCS activities, it is important to build one which favours as much as possible the schedulability degree of FPS and EDF activities.

In Section 5.1.1 we present the schedulability analysis for a set of FPS and EDF tasks and messages, considering a fixed given static schedule for SCS activities. In Section 5.1.2 we present the actual holistic scheduling algorithm which constructs the static schedule and is driven by the objective of achieving a global schedulability of the system.

5.1.1. Response Time Analysis

In order to keep the presentation reasonably simple and given the space limitations, we present here the analysis for a restricted model in the sense that SCS tasks are communicating only through TT phases, while the communication among FPS and EDF tasks is only through ET phases. This is not an inherent limitation of our approach, and the analysis we have developed and implemented supports the general model.

A FPS or EDF task is activated by a scheduled event. Each task $t_{ij}$ has an offset $\phi_{ij}$, which specifies the earliest activation time of $t_{ij}$ relative to the occurrence of the triggering event. The delay between the earliest possible activation time of $t_{ij}$ and its actual activation time is modelled as a jitter $J_{ij}$ (Figure 3). Offsets and jitters are the means by which dependencies among tasks are modelled for the schedulability analysis. The response time $R_{ij}$ of a task $t_{ij}$ is the time measured from the occurrence of the associated event until the completion of $t_{ij}$. Each task $t_{ij}$ has a best case response time $R_{ij}^{\text{best}}$.

Figure 3. The General Strategy

In order to determine if a hierarchically scheduled system is schedulable, we used as a starting point the schedulability analysis algorithm for EDF-within-FPS systems, developed in [13]. In this section, we present our extension to this algorithm, which allows us to compute the worst case response times for the FPS and EDF activities when they are interfered by the SCS activities.

In [13] the authors have developed a schedulability analysis algorithm for EDF tasks running under a hierarchical FPS/EDF scheduling policy. Response times for the tasks are obtained using workload equations:

- For FPS tasks, the worst case response times are influenced only by higher priority tasks, so the completion time of an activation $p$ of task $t_{ab}$ is given by the following recursion:
  \[ w_{ab}^k+1(p) = b_{ab}^p + c_{ab}^p + \sum_{\forall p_i \in \mathcal{P}_{pri}^{ab}} \left[ w_{ab}^k(p_i) + J_{ij} \right] \]

  where $p$ is the number of activations of $t_{ab}$ in the busy period, $b_{ab}$ is the activation time for $t_{ab}$ (see [13] for a discussion regarding the blocking time), and $J_{ij}$ and $T_{ij}$ are the maximum release jitter and the period of $t_{ij}$, respectively. The worst case response time $R_{ab}^p$ is then computed as the maximum for all possible values of $R_{ab}^p = w_{ab}^k(p) - (p - 1)T_{ab} + J_{ij}$.

- For EDF tasks, the worst case response times are influenced by higher priority tasks and by EDF tasks running at the same priority level as the task under analysis:
  \[ w_{ab}^k(p) = b_{ab}^p + c_{ab}^p + \sum_{\mathcal{P}_{pri}^{ab} \neq \mathcal{P}_{pri}^{ij}} W_{ij}(w_{ij}^k(p)) + D_{ij}(p) \]

  where $W_{ij}(T_{ij}) = \left[ \frac{T_{ij}}{T_{ij}} \right] C_{ij}$ and $D_{ij}(p)$ is the deadline of activation number $p$, when the first activation of $t_{ab}$ occurs at time $A$:

  \[ D_{ab}(p) = A - J_{ab}^p - (p - 1)T_{ab} + D_{ab} \]

  The analyzed intervals are extracted from situations in which the task under analysis $t_{ab}$ has the deadline larger or equal than the deadline of the other tasks in the system. The worst case response time $R_{ab}^p$ for a task running under EDF-within-FPS is then computed as the maximum for all possible values of $R_{ab}^p = w_{ab}^k(p) - (p - 1)T_{ab} + J_{ij} - A$

A similar technique is used in the more complex case of offset-based analysis. However, regardless of the analysis used, the technique has to be enhanced to take into consideration an existing static schedule, allowing us to analyze hierarchically scheduled systems that use a combination of SCS, FPS and EDF scheduling policies.

Our extension takes into consideration the interference produced by an existing static schedule when computing the worst-case response times of FPS and EDF activities scheduled using EDF-within-FPS.

First we introduce the notion of ET demand associated with an FPS or EDF task $t_{ij}$ on a time interval $t$ as the maximum amount of CPU time or bus time which can be demanded by higher or equal priority ET activities and by $t_{ij}$ during the time interval $t$. In Figure 4, the ET demand of the task $t_{ij}$ during the busy window $w_{ij}$ is denoted with $H_{ij}(t)$, and it is the sum of the worst case execution times for task $t_{ij}$ and two other higher priority tasks $t_{ab}$ and $t_{cd}$. During the same interval $t$, we define the availability as the processing time which is not used by statically scheduled activities. In Figure 4, the CPU availability for the analyzed interval is obtained by subtracting from the amount of processing time needed for the SCS activities.

During a busy window $w_{ij}$, the ET demand $H_{ij}^p$ associated with the task under analysis $t_{ij}$ is equal with the length of the busy window which would result when considering only ET activity on the system:

\[ H_{ij}^p(t) = w_{ij}^p(t) \]

During a time interval $t$, the availability $A_{ij}$ associated with task $t_{ij}$ is:

\[ A_{ij}^p(t) = \min[A_{ij}^p(t)] \]

where $A_{ij}^p(t)$ is the total available CPU-time on processor $M(t_{ij})$ in the interval $[\phi_{ij}, \phi_{ij} + t]$, and $\phi_{ij}$ is the start time of task $t_{ij}$, as recorded in the static schedule table.
The discussion above is, in principle, valid for both types of ET tasks (i.e., FPS and EDF tasks) and messages. However, there exist two important differences. First, messages do not preempt each other, therefore, in the demand equation the blocking term will be non-zero, equal with the largest transmission time of any ET message. Second, the availability for a message is computed by subtracting from t the length of the ST slots which appear during the considered interval; moreover, because an ET message will not be sent unless there is enough time before the current dynamic phase ends, the availability is further decreased by a SCS task 1, a static phase in the busy window (where 1 is the transmission time of the longest ET message).

Our schedulability analysis algorithm determines the length of a busy window for FPS and EDF tasks and messages by identifying the appropriate size of w, for which the ET demand is satisfied by the availability: H(w(t)) ≤ A(w(t)). This procedure for the calculation of the busy window is included in the iterative process for calculation of response times presented in Section 5.1.2. It is important to notice that this process includes both tasks and messages and, thus, the resulted response times of the FPS and EDF tasks are computed by taking into consideration the delay induced by the bus communication.

5.1.2. Static Scheduling

As mentioned in the beginning of Section 5.1, building the static cyclic schedule for the SCS activities in the system has to be performed in such a way that the interference imposed on the FPS and EDF activities is minimum. The holistic scheduling algorithm is presented in Figure 6. The construction of the schedule table with start times for SCS tasks and messages, we adopted a list scheduling-based algorithm [4] which iteratively selects tasks and schedules them appropriately.

A ready list contains all SCS tasks and messages which are ready to be scheduled (they have no predecessors or all their predecessors have been scheduled). From the ready list, tasks and messages are extracted one by one (Figure 6, line 2) to be scheduled on the processor they are mapped to, or into a static bus-slot associated to that processor on which the sender of the message is executed, respectively. The priority function which is used to select among ready tasks and messages is a critical path metric, modified for the particular goal of scheduling tasks mapped on distributed systems. Let us consider a particular task 2 selected from the ready list to be scheduled. We consider that ASAP and is the earliest time moment which satisfies the condition that all preceding activities (tasks or messages) of 2 are finished and processor 1 is free. The moment 2AP is the latest time when 2 can be scheduled. With only the SCS tasks in the system, the straightforward solution would be to schedule 2 at ASAP. In our case, however, such a solution could have negative effects on the schedulability of FPS and EDF tasks.

In order to consider only a limited number of possible positions for the start time of a SCS task, we take into account the information obtained from the schedulability analysis described in Section 5.1.1, which allows us to compute the response times of ET (i.e., FPS and EDF tasks). We start from the observation that statically scheduling a SCS task 2 so that the length of busy-period of an ET activity is not modified will consequently lead to unchanged worst-case response time for that ET task. This can be achieved by providing for enough available processing time between statically scheduled tasks so that the busy period of the ET task does not increase. For example, in Figure 5.a we can see how statically scheduling two SCS tasks 1 and 2 influences the busy period w, of a FPS (or EDF) task 3. Figure 5.a1 presents the system with only 3 scheduled, the system with the busy-period w is computed. Figure 5.a2 shows how scheduling another SCS task 2 too early decreases the availability during the interval [a1, a1 + w3], and consequently leads to an increase of w3 and R3, respectively. Such a situation is when the two SCS tasks are scheduled like in Figure 5.a3, where no extra interference is introduced in the busy period w3. However, during the static scheduling, we have to consider two aspects: the interference on the timeline during the busy period w3 is computed by taking into consideration the delay induced by the bus communication.

Figure 5. Static Scheduling

The interference on the timeline during the busy period w3 is computed by taking into consideration the delay induced by the bus communication.

Figure 6. Holistic Scheduling Algorithm

1. The interference with the FPS and EDF activities should be minimized;
2. The deadlines of TT activities should be satisfied.

The technique presented in Figure 5.a takes care only of the first aspect, while ignoring the second. One may notice that scheduling a SCS task later increases the probability that we will not be able to find feasible start times for that particular task or for the SCS tasks which depend on it and are not scheduled yet (for example, in Figure 5.b1, task 2 misses its deadline and the resulted schedule is invalid). We reduce such a risk by employing the technique presented in Figure 5.b2-b3, where we first schedule the second task so that we maximize the continuous slack between the jobs of tasks 1 and 2; for this reason, we place 2 in the middle of the slack between the last SCS task in the first period of the static schedule (the first job of task 1), and the first task scheduled in the second period (the second job of task 1). In such a situation, the maximum busy period w, of the ET tasks may increase due to interference from task 2 (Figure 5.b2). However, considering that such an increase is acceptable (in the sense that no ET tasks miss their deadlines), then we can now improve the probability of finding a valid static scheduling by scheduling the task 2 earlier in time, as long as the maximum ET busy period wmax does not increase (Figure 5.b3).

The scheduling algorithm is presented in Figure 6. If the selected SCS activity extracted from the ready_list is a task 2, then the task is first scheduled in the middle of the slack at the end of the period T2 of the static schedule (line 10). In order to determine the response times of the ET activities and the maximum busy period wmax in the resulted system, the scheduled application is analyzed using the technique in Section 5.1.1 (line 11). The value obtained for wmax is then used for determining how early the task 2 can be scheduled without increasing the response times of the ET tasks (line 12). When scheduling a ST message extracted from the ready list, we place it into the first bus-slot associated with the sender node in which there is sufficient space available (line 6). If all SCS tasks and messages have been scheduled and the schedulability analysis for the ET tasks indicates that all ET activities meet their deadlines, then the global system scheduling has succeeded.

5.2 Mapping & Scheduling Policy Assignment Heuristic

In Step 2 of our optimization strategy (Figure 3), the following design transformations are performed with the goal to produce a schedulable system implementation:

- change the scheduling policy of a task,
- change the mapping of a task,
- change the priority level of a FPS of EDF task.

Our optimization algorithm is presented in Figure 7 and it implements a greedy approach in which every task in the system is iteratively mapped on each node (line 4) and assigned to each scheduling policy (line 8), under the constraints imposed by the designer. The next step involves adjustments to the bus access cycle (line 10), which are needed for the case when the bus cycle configuration cannot handle the minimum requirements of the current internode.
communication. Such adjustments are mainly based on enlargement of the static slots or dynamic phases in the bus cycle, and are required in the case the bus has to support larger messages than before. New messages may appear on the bus due to, for example, remapping of tasks. For more details on the subject of bus access optimization and adjustment, the reader is referred to [16].

Before the system is analyzed for its timing properties, our heuristic also tries to optimize the priority assignment of tasks running under FPS (line 11). The state of the art approach for such a task is the HOPA algorithm for assigning priority levels to tasks in multiprocessor systems [8]. However, due to the fact that HOPA is computationally expensive to be run inside such a design optimization loop, we use a scaled down greedy algorithm, in which we drastically reduce the number of iterations needed for determining an optimized priority assignment.

Finally, the resulted system configuration is analyzed (line 15) using the scheduling and schedulability analysis algorithm presented in Section 5.1. The resulted cost function will decide whether the current configuration is better than the current best one (lines 16–19). Moreover, if all activities meet their deadlines $O_A < 0$, the optimization heuristic stops the exploration process and returns the current best-so-far configuration (lines 20–22).

6. Experimental Results

For the evaluation of our design optimization heuristic we have used synthetic applications as well as a real-life example consisting of a vehicle cruise controller. Thus, we have randomly generated applications of 40, 60, 80 and 100 tasks on systems with 4 processors. 56 applications were generated for each dimension, thus a total of 224 applications were used for experimental evaluation. An equal number of applications with processor utilizations of 20%, 40%, 60% and 80% were generated for each application dimension. All experiments were run on an AMD AthlonXP 2400+ processor, with 512 MB RAM.

We were first interested to determine the quality of our design optimization approach for hierarchically scheduled systems, the MSPA heuristic (MSPA); see Figure 7. We have compared the percentage of schedulable implementations found by MSPA with the number of schedulable solutions obtained by the InitialMSPA algorithm described in Figure 5 (line 2), which derives a straightforward system implementation, denoted with SF. The results are depicted in Figure 8.a. We can see that our MSPA heuristic (the black bars) performs very consistently higher than the number of schedulable solutions obtained with the SF approach (the white bars). On average, MSPA finds 44.5% more schedulable solutions than SF.

Second, we were interested to determine the impact of the scheduling policy assignment (SPA) decisions on the number of schedulable applications obtained. Thus, for the same applications, we considered that the task mapping is fixed by the SF approach, and only the SPA is optimized. Figure 8.a presents this approach, labelled “MSPA/No mapping”, corresponding to the gray bars. We can see that most of the improvement over the SF approach is obtained by carefully optimizing the SPA in our MSPA heuristic.

We were also interested to find out what is the impact of the processor utilization of an application on the quality of the implementations produced our optimization heuristic. Figure 8.b presents the percentage of schedulable solutions found by MSPA and SF as we ranged the utilization from 20% to 80%. We can see that SF degrades very quickly with the increased utilization, with under 10% schedulable solutions for applications with 40% utilization and without finding any schedulable solution for applications with 80% utilization, while MSPA is able to find a significant number of schedulable solutions even for high processor utilizations.

Finally, we considered a real-life example implementing a vehicle cruise controller (CC). The process graph that models the CC has 32 processes, and is described in [15]. The CC was mapped on an architecture consisting of three nodes: Electronic Throttle Module (ETM), Anti-lock Breaking System (ABS) and Transmission Control Module (TCM). We have considered a deadline of 250 ms. In this setting, SF failed to produce a schedulable implementation. Our design optimization heuristic MSPA was considered first such that the mapping is fixed by SF, and we only allowed reassigning of scheduling policies. After 25.5 seconds, the best scheduling policy allocation which fulfilled all constraints produced a “degree of schedulability” three times higher than obtained by SF. When mapping was allowed, and a schedulable system was found after 28.49 seconds.

7. Conclusions

In this paper we have addressed the analysis and optimization of hierarchically scheduled heterogeneous real-time systems. Several scheduling policies are used for tasks, such as static cyclic scheduling, fixed-priority preemptive scheduling and earliest deadline first, organized as a hierarchy. Messages are transmitted using the Universal Communication Model that combines both time-triggered and event-triggered slots.

We have proposed a holistic scheduling analysis that is able to handle the hierarchical scheduling policy allocation problem. For high processor utilization, we have proposed a design optimization heuristic for the assignment of scheduling policies to tasks, the mapping of tasks to hardware components, and the scheduling of the activities such that the timing constraints of the application are guaranteed.

As our experiments have shown, our heuristic is able to find schedulable implementations under limited resources, achieving an efficient utilization of the system.

References