ANALYSIS AND OPTIMIZATION OF MIXED-CRITICALITY APPLICATIONS ON PARTITIONED DISTRIBUTED ARCHITECTURES

D. Tămaş-Selicean, S. O. Marinescu and P. Pop
Technical University of Denmark, Department of Informatics, 2800 Kongens Lyngby, Denmark
dota@imm.dtu.dk

Abstract

In this paper we are interested in mixed-criticality applications implemented using distributed heterogenous architectures, composed of processing elements (PEs) interconnected using the TTEthernet protocol. At the PE-level, we use partitioning, such that each application is allowed to run only within predefined time slots, allocated on each processor. At the communication-level, TTEthernet uses the concepts of virtual links for the separation of mixed-criticality messages. TTEthernet integrates three types of traffic: Time-Triggered (TT) messages, transmitted based on schedule tables, Rate Constrained (RC) messages, transmitted if there are no TT messages, and Best Effort (BE) messages. We assume that applications are scheduled using Static Cyclic Scheduling (SCS) or Fixed-Priority Preemptive Scheduling (FPS). We are interested in analysis and optimization methods and tools, which decide the mapping of tasks to PEs, the sequence and length of the time partitions on each PE and the schedule tables of the SCS tasks and TT messages, such that the applications are schedulable and the response times of FPS tasks and RC messages is minimized. We have proposed a Tabu Search-based meta-heuristic to solve this optimization problem, which has been evaluated using several benchmarks.

1 Introduction

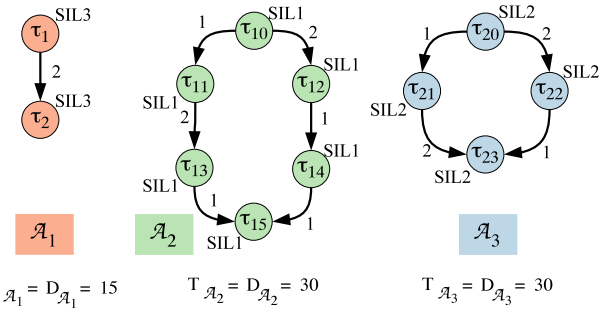
Safety is a property of a system that will not endanger human life or the environment. *Safety-Integrity Levels* (SILs) capture the required protection against failure when building a safety-critical embedded system, and will dictate the development processes and certification procedures that have to be followed. The current trend is towards *integrated architectures*, where applications of different safety criticality levels are integrated into the same platform. Such *mixed-criticality* applications can be integrated onto the same architecture only if there is enough spatial and temporal separation among them.

In this paper we address mixed-criticality hard real-time applications mapped on heterogeneous distributed architectures. We consider that the computation-level temporal and spatial separation mechanisms are provided by "Integrated Modular Avionics" (IMA) [12] through the concept of *partitions*. Similar separation mechanisms are available in other industries [7], [9]. With IMA, each application runs in a separate partition on the processing elements (PEs) where the application is mapped.

At the communication level, separation is provided by the TTEthernet protocol [2] protocol, which uses the concept of *virtual links* for the separation of mixed-criticality messages. TTEthernet is a deterministic, synchronized and congestion-free network based on the IEEE 802.3 Ethernet standard and integrates three types of traffic. Time-Triggered (TT) messages, Event-Triggered (ET) messages with bounded end-to-end delay, also called Rate Constrained (RC) messages, and Best-Effort (BE) messages, for which no timing guarantees are given. The TT frames are transmitted based on static schedule tables, and have the highest priority. RC messages are transmitted if there are no TT messages, while BE traffic has the lowest priority.

We assume that applications are scheduled using static-cyclic scheduling (SCS) or fixed-priority scheduling (FPS). Applications scheduled with SCS transmit TT frames, while the ones scheduled with FPS transmit the messages as RC frames. In this paper, we propose a tool for the analysis and optimization of mixed-criticality applications on partitioned architectures. The tool takes as input the model of the application, consisting of tasks and messages, the model of the system platform, and produces that system configuration which guarantees the separation of mixed-criticality applications and their schedulability. The analysis calculates the worst-case response times of FPS tasks and the worst-case transmission times of RC messages, taking into account the particularities of IMA and TTEthernet, respectively. The optimization decides the mapping of tasks to the processing elements (PEs) of the architectures, the sequence and length of the time-partitions on each PE and the schedule tables of the SCS tasks and TT messages, such that the applications are schedulable and, in addition, the response times of FPS tasks and RC messages is minimized.

Researchers have recently started to address the integration of mixed-criticality tasks onto the same platform [4], [14]. In [14], we proposed an optimization approach to determine the mapping of tasks to PEs, the time partitions on each PE



(b)

	τ1	τ2	τ10	τ11	τ12	τ13	τ14	τ15	τ20	τ21	τ22	τ23
N1	x	3	2	3	x	2	6	2	4	6	4	x
N2	4	5	3	x	6	3	9	6	9	10	5	4

Fig. 1: Application model example

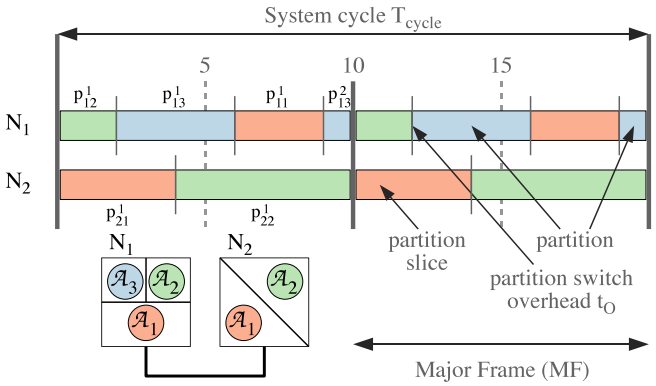


Fig. 2: Partitioned architecture

and the schedule tables, such that all applications are schedulable and the development costs are minimized. In that work, communications were ignored, and a simple unpartitioned statically scheduled shared bus was used.

2 System Model

The set of all applications in the system is denoted with Γ. We model an application as a directed, acyclic graph $\mathcal{G}_i(\mathcal{V}_i, \mathcal{E}_i) \in \Gamma$. Each node $\tau_j \in \mathcal{V}_i$ represents one task. The mapping is denoted by the function $M : \mathcal{V}_i \rightarrow \mathcal{N}$, where $\mathcal{N}$ is the set of PEs in the architecture. For each task τ_i we know the worst-case execution time (WCET) $C_i^{N_j}$ for each PE N_j where τ_i is considered for mapping. An edge $e_{jk} \in \mathcal{E}_i$ from τ_j to τ_k indicates that the output of τ_j is the input of τ_k. Communication between tasks mapped to different PEs is performed by message passing on the TTEthernet network. Applications are scheduled using SCS or FPS. A deadline $D_{\mathcal{G}_i} \leq T_{\mathcal{G}_i}$, where $T_{\mathcal{G}_i}$ is the period of $\mathcal{G}_i$, is imposed on each graph $\mathcal{G}_i$.

An example mixed-criticality system composed of three applications is presented in Fig. 1a. The WCETs of tasks are given in Fig. 1b for two PEs, N_1 and N_2. An "x" in the table means that the task is not considered for mapping on the respective PE. The size of the messages is depicted on the graph edges.

2.1 Safety Integrity Levels

During the engineering of a safety-critical system, the hazards are identified and their severity is analyzed, the risks are assessed and the appropriate risk control measures are introduced to reduce the risk to an acceptable level. A Safety-Integrity Level (SIL) captures the required level of risk reduction. SILs are assigned to safety functions, from SIL 4 (most critical) to SIL 0 (non-critical). Functions are decomposed into tasks. We introduce the notation $SIL : \mathcal{V}_i \rightarrow \{SIL\ k\}$, where $k \in \{0..4\}$, to capture the SIL of a task. The tasks of an application may have different SILs. The SILs for the example in Fig. 1a are presented next to the tasks.

Tasks of different SILs have to be separated. Otherwise, for example, a lower-criticality task could write in the code or data area of a higher-criticality task, leading thus to a failure. The same is true for mixed-criticality messages.

2.2 Separation at PE-level

We consider architectures composed of a set $\mathcal{N}$ of PEs connected via TTEthernet. We denote the assignment of tasks to partitions using the function $\phi : \mathcal{V} \rightarrow \mathcal{P}$, where $\mathcal{V}$ is the set of tasks in the system and $\mathcal{P}$ is the set of partitions. On a PE N_i, a partition $P_j \in \mathcal{P}$ is defined as the sequence $\mathcal{P}_{ij}$ of k partition slices p_{ij}^k, $k \geq 1$. A partition slice p_{ij}^k is a predetermined time interval in which the tasks mapped to N_i and allocated to the partition P_j are allowed to use N_i. All the slices on a processor are grouped within a Major Frame (MF), that is repeated periodically. The period T_{MF} of the major frame is given by the designer and is the same on each PE. Several MFs are combined together in a system cycle that is repeated periodically, with a period T_{cycle}.

Fig. 2 presents the partitions for 3 applications of different SILs, $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$, implemented on an architecture of 2 PEs, N_1 and N_2, with $T_{MF} = 10$ and $T_{cycle} = 2 \times T_{MF} = 20$.

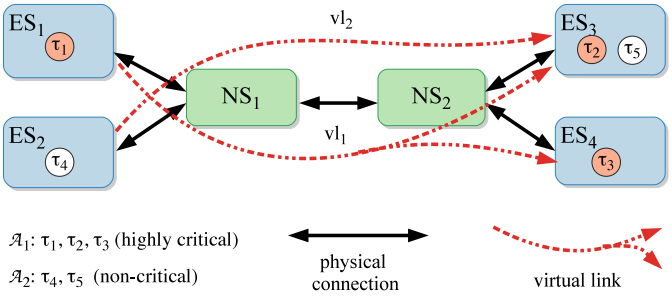


Fig. 3: TTEthernet cluster example

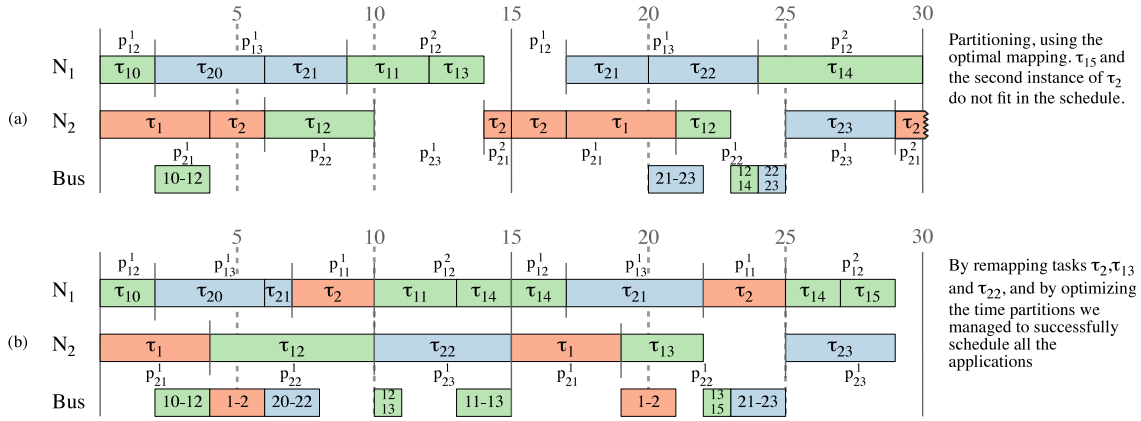


Fig. 4: Motivational example

2.3 Separation at Network-level

A TTEthernet network is composed of a set of End Systems (ESes) interconnected by links and Network Switches (NSes). An example is presented in Fig. 3, where we have 4 ESes, ES_1 to ES_4, and 2 NSes, NS_1 and NS_2.

We model a TTEthernet network as an undirected graph $\mathcal{G}_N(\mathcal{V}_N, \mathcal{E}_N)$, where $\mathcal{V}_N = \mathcal{ES} \cup \mathcal{NS}$ is the set of end systems ($\mathcal{ES}$) and network switches ($\mathcal{NS}$) and $\mathcal{E}_N$ is the set of physical links. For Fig. 3, $\mathcal{V}_N = \mathcal{ES} \cup \mathcal{NS} = \{ES_1, ES_2, ES_3, ES_4\} \cup \{NS_1, NS_2\}$, and the physical links $\mathcal{E}_N$ are depicted with thick, black, double arrows.

The space partitioning between messages of different criticality transmitted over physical links and network switches is achieved through the concept of *virtual link*. Virtual links connect one sender to multiple receivers. Each virtual link carries a single message.

We denote the set of virtual links in a cluster with $\mathcal{VL}$. A virtual link $vl_i \in \mathcal{VL}$ is a directed tree, with the sender as the root and the receivers as leafs. Each virtual link is composed of a set of dataflow paths, one such dataflow path dp_i for each root-leaf connection. For example, in Fig. 3, $vl_1 = dp_1 \cup dp_2$, and dp_1 connects ES_1 to ES_3, while dp_2 connects ES_1 to ES_4. A *dataflow link* $l_i = [\nu_j, \nu_k] \in \mathcal{L}$, where $\mathcal{L}$ is the set of dataflow links, is a directed communication connection from ν_j to ν_k, where ν_j and $\nu_k \in \mathcal{V}$ can be ESes or NSes. Using this notation, a dataflow path such as dp_1 in Fig. 3 can be denoted as $[[ES_1, NS_1], [NS_1, NS_2], [NS_2, ES_3]]$.

We assume that the topology of virtual links and the traffic classes for each frame are given by the designer, and we define the sets $\mathcal{F}^{TT}$, $\mathcal{F}^{RC}$ and $\mathcal{F}^{BE}$, respectively, with $\mathcal{F} = \mathcal{F}^{TT} \cup \mathcal{F}^{RC} \cup \mathcal{F}^{BE}$.

The size $f_i.size$ for each frame $f_i \in \mathcal{F}$ is given. In addition, for the TT and RC frames we know their periods and deadlines, $f_i.period$ and $f_i.deadline$, respectively. We define the rate of an RC frame f_i as $f_i.rate = 1/f_i.period$. Knowing the size of a frame f_i and the given speed of a dataflow link $[\nu_j, \nu_k]$, we can determine the transmission duration $C_i^{[\nu_j,\nu_k]}$ of f_i on $[\nu_j, \nu_k]$.

3 Problem Formulation

The problem we are addressing in this paper can be formulated as follows: given a set Γ of applications, the criticality level $SIL(\tau_i)$ of each task τ_i, an architecture consisting of a set $\mathcal{N}$ of processing elements, the topology of the TTEthernet network $\mathcal{G}_N$, the set of TT and RC frames $\mathcal{F}^{TT} \cup \mathcal{F}^{RC}$, the set of virtual links, the size of the major frame T_{MF} and the application cycle T_{cycle}, we are interested to find an implementation Ψ such that all applications meet their deadlines and the response times of FPS tasks and RC messages are minimized. Deriving an implementation Ψ means deciding on the mapping M of tasks to PEs, the set $\mathcal{P}$ of partition slices on each processor, including their order and size, the assignment ϕ of tasks to partitions and the schedule $\mathcal{S}$ for all the tasks and the schedules $\mathcal{S}_{TT}$ for the TT frames.

3.1 Mapping and Partition Optimization

Let us illustrate the problem of optimizing the mapping of tasks and partitioning of PEs using the mixed-criticality applications $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$ from Fig. 1a, to be implemented on two PEs, N_1 and N_2. We have set T_{MF} to 15 time units and $T_{cycle} = 2 \times T_{MF} = 30$. In this example we use a simple static bus.

Let us assume that we have determined the optimal mapping of tasks, in terms of the cost function from Section 4, ignoring the partitioning. Using this optimal mapping, we are interested to obtain the partitions and the schedules, such that, the separations are enforced and the schedule lengths are minimized with the goal of producing a schedulable implementation. This partitioning is presented in Fig. 4a. In this case, although application $\mathcal{A}_3$ is schedulable, task τ_{15} and the second instance

of task τ_2 do not fit into the schedule, and thus applications $\mathcal{A}_1$ and $\mathcal{A}_2$ are not schedulable.

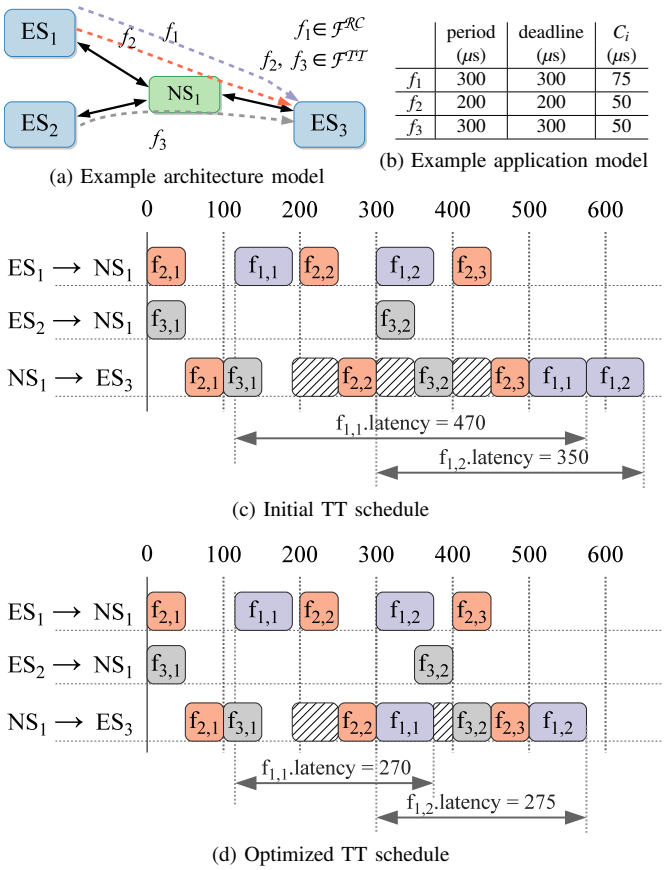
In this paper we consider that the optimization of mapping and partitioning is done at the same time, and not separately. By deciding simultaneously the mapping and partitioning we have a better chance of obtaining schedulable implementations. Such a solution is depicted in Fig. 4b, where all applications are schedulable. Compared to the solution in Fig. 4a, we have changed the mapping of tasks τ_{13} and τ_{22} from N_1 to N_2 and of task τ_2 from N_2 to N_1, and we have resized the partition slices and changed the schedule accordingly.

3.2 Optimization of TT Message Schedules

Let us illustrate the TT frame schedule synthesis problem using the setup from Fig. 5, where we have a network composed of three ESes, ES_1 to ES_3 and a network switch NS_1 (see Fig. **??**) and three frames, see the table in Fig. **??**. The details of the virtual links and frames are in the table and figure. The dataflow links have the same speed. For this example we consider that the RC and TT traffic are integrated using a "timely block" policy, i.e., an RC frame will be delayed if it could block a scheduled TT frame.

We want to determine the TT schedules $\mathcal{S}_{TT}$ such that all the TT and RC frames are schedulable. The schedulability of a TT frame f_i is easy to determine: we just have to check the schedules $\mathcal{S}_{TT}$. To determine the schedulability of an RC frame f_j we have to compute its worst-case end-to-end delay, from the moment it is sent to the moment it is received. We denote this worst-case delay with R_{f_j}. Fig. 5 presents two possible solutions for synthesizing the TT schedules $\mathcal{S}_{TT}$. In both cases, Fig. 5a and 5b, we show a Gantt chart, which shows on a timeline from 0 to 600 μs what happens on the three dataflow links, $[ES_1, NS_1]$, $[ES_2, NS_1]$ and $[NS_1, ES_3]$.

Because we are interested in the schedulability of RC frames, for the RC frame f_1 we show in both cases (c) and (d) in Fig. 5 the worst-case scenario, i.e., the situation which has generated the largest (worst-case) end-to-end delay. In Fig. 5a the TT schedules are constructed such that the end-to-end delay of TT frames is minimized, i.e., the TT frames arrive at their destination as soon as possible. In this case, the worst-case end-to-end delay of the RC frame f_1, namely R_{f_1}, is 470 μs, which is greater than its deadline of 300 μs, hence f_1 is not schedulable. This worst-case for f_1 happens for the first frame instance $f_{1,1}$, see Fig. 5a, when $f_{1,1}$ happens to be sent by ES_1 at 105 μs. In this case, as the network implements the *timely block* integration algorithm, the frame cannot be forwarded by NS_1 to ES_3 until there is a big enough time interval to transmit the frame without disturbing the scheduled TT frames. We denote these "blocked" time intervals with hatched boxes. The first big enough interval starts only at time 500, right after $f_{2,3}$ is received by ES_3, which is too late.



(a) Example architecture model

(b) Example application model

(c) Initial TT schedule

(d) Optimized TT schedule

Fig. 5: Worst-case scenario for RC frame f_1

However, if we instead schedule the TT frame f_3 such that its second instance $f_{3,2}$ will be sent by ES_2 to NS_1 at 350 μs, the worst case end-to-end delay for f_1 is reduced to 275, hence f_1 is schedulable. Such a solution is depicted in Fig. 5b, where we also depict the worst-case scenario for f_1.

This example shows that by considering the RC traffic when scheduling the TT frames, the impact of the TT schedule on the latency of the RC frames can be greatly reduced.

4 Mixed-Criticality System Optimization

The problem presented in the previous section is NP-complete, hence we use a meta-heuristic called Tabu Search (TS) to solve it. Starting from an initial solution, we iteratively run (i) a TS meta-heuristic to optimize the task mapping and the set of partition slices, while considering the schedules for TT frames to be fixed, and (ii) a TS-based TT frames schedule optimization algorithm considering the previously obtained task mapping and task partition slices as fixed.

4.1 Tabu Search

TS [8] is a meta-heuristic optimization, which searches for that solution which minimizes the *cost function* of an imple-

mentation Ψ:

$$Cost(\Psi) = \begin{cases} c_1 = \sum_{\mathcal{A}_i \in \Gamma} \max(0, R_i - D_i) & \text{if} c_1 > 0 \\ c_2 = \sum_{\mathcal{A}_i \in \Gamma} (R_i - D_i) & \text{if} c_1 = 0 \end{cases} \quad (1)$$

R_i is the response time of the application, while D_i is the deadline of the application. If at least one application is not schedulable, there exists one R_i greater than the deadline D_i, and therefore the term c_1 will be positive. However if all the applications are schedulable, this means that each R_i is smaller than D_i, and the term $c_1 = 0$. In this case, we use c_2 as the cost function, since the applications are schedulable, we are interested to minimize response time of the applications.

The worst-case response times for SCS tasks and TT messages are determined based on the schedule tables, while those of FPS tasks and RC messages are calculated using the analysis from Section 4.2.

Tabu Search explores the design space by using design transformations (or "moves") applied to the current solution in order to generate neighboring solutions. To escape local minima, TS incorporates an adaptive memory (called "Tabu list"), to prevent the search from revisiting previous solutions, thus avoiding cycling.

At the PE level, we use one *re-assignment* move, which changes the assignment of a task to another partition and four types of moves applied to partition slices: *resize*, *swap*, *join* and *split*. The *re-assignment* move re-assigns a task to another, randomly chosen, partition. The partition can be on another PE, thus, implicitly, the re-assignment move will also *re-map* the task. The *resize* move, resizes the selected partition slice. The *swap* move swaps the chosen partition slice with another randomly chosen partition slice. The *join* move joins two partition slices belonging to the same application, while the *split* move splits a partition slice into two, and adds the second slice to the end of the MF.

For the TT frames, we use four types of moves: *advance*, *advance predecessors*, *postpone* and *postpone successors*. The *advance* move will advance the scheduled send time of a TT frame instance $f_{i,x}$ from a node ν_j on a dataflow link $[\nu_j, \nu_k]$ to an earlier moment in time. The *advance predecessors* applied to a frame instance $f_{i,x}^{[\nu_j,\nu_k]}$, will advance the scheduled send time for all its predecessors. Similarly the *postpone* move will postpone the schedule send time of a TT frame instance from a node, while *postpone successors* will postpone the send time for all the successors of that frame instance.

4.2 Analysis of Mixed-Criticality Applications

To determine the schedulability of FPS tasks we use a response-time analysis [5] to calculate the worst-case response time R_i of every FPS task τ_i, which is compared to its deadline D_i. Audsley and Wellings [3] have proposed a schedulability analysis for FPS tasks using temporal partitioning (IMA), which, when analyzing a FPS task in a certain partition, considers the other time-partitions as higher priority tasks. This analysis assumes that the deadlines are smaller or equal to the periods, that the tasks are independent, and that the start times of partition slices within a major frame are periodic. In this paper we use the analysis proposed by us in [10] for FPS tasks.

The worst-case end-to-end delay R_{f_i} of an RC frame $f_i \in \mathcal{F}^{RC}$ sent on a virtual link $vl_i = \mathcal{M}(f_i)$ is the sum of the worst-case queueing delays $Q_{f_i}^{[\nu_j,\nu_k]}$ on each network node (ES or NS) $\nu_j \in \mathcal{V}$ (which is the source of a dataflow link $[\nu_j, \nu_k] \in vl_i$) and the transmission duration $C_{f_i}^{[\nu_j,\nu_k]}$ for each dataflow link $[\nu_j, \nu_k] \in vl_i$ the frame transits. For RC frames, we use the analysis from [13].

5 Experimental Results

Regarding tasks, we were interested to evaluate the proposed optimization strategy (TO) in terms of its ability to find schedulable implementations. Thus, we have used 5 synthetic benchmarks with 4 to 6 mixed-criticality applications mapped on 2 to 6 PEs. Together with TO, Table I also presented the results obtained using a Straightforward Solution for tasks (SST). SST consists of a simple straight forward partitioning scheme which allocated for each application $\mathcal{A}_j$ a total time on PE $_i$ proportional to the utilization of the tasks of $\mathcal{A}_j$ mapped on N_i. The partitions P_{ij} thus allocated have the same length and they are distributed with a period equal to the smallest period of a task from $\mathcal{A}_j$ mapped to N_i. Column 2 in Table I presents the number of tasks. The number of schedulable tasks (out of the total) obtained by our proposed TO strategy is presented in column 5, while column 4 presents the results obtained using SST.

As we can see from "Set 1", SST which does not perform optimization, is not able to find schedulable implementations. However, by applying our proposed TO approach, we are able to optimize the time partitions such that all applications are schedulable. We have measured the ability of TO to improve over SST by using a percentage average increase in the degree of schedulability over all applications, presented in the last column. As we can see there is a dramatic increase over all applications, when using TO. This means that we can potentially implement the applications on a slower (cheaper) architecture.

We have also used 2 real life benchmarks derived from the Embedded Systems Synthesis Benchmarks Suite (E3S) version 0.9 [6], see "Set 2". We have used the *telecom-mocsyn* and *auto-indust-cowls* benchmarks. The applications are mapped on an architecture of 3 PEs. The results obtained from these real-life benchmarks confirm the results of the synthetic benchmarks.

Set	Tasks	PEs	SST Sched. Tasks	TO Sched. Tasks	avg. % increase in δ
1	20	2	10	All	832.88
	26	3	13	All	27.36
	40	4	6	All	88.41
	50	5	10	All	73.57
	62	6	26	All	278.72
2	24	3	5	All	113.95
	25	3	All	All	61.87

TABLE I: Experimental results for tasks

Set	Test case	ES	NS	Messages	Frame instances	Δ_{cost} [%]
1	11	13	4	80	12593	2.58
	12	25	6	88	1787	24.44
	13	35	8	103	2285	20.06
	14	45	10	165	3299	11.90
2	21	11	4	115	16904	9.17
	22	25	6	179	2523	20.61
	23	35	8	154	3698	39.34
3	automotive	15	3	170	38305	50.88

TABLE II: Experimental results for messages

The results related to messages are presented in Table II. For the synthetic benchmarks, we have used 6 network topologies, and we have randomly generated the parameters for the frames, taking into account the details of the TTEthernet protocol. For all experiments, we have compared our optimization strategy for messages (let us call it TM) with a baseline solution, namely the Straightforward Solution for messages (SSM), which builds the TT schedules with the goal of minimizing the end-to-end response time of the TT frames without considering the RC traffic. The comparison between SSM and TM, Δ_{cost}, is shown in the last column in the table as a percentage improvement of TM over SSM.

In the sets of experiments labeled "Set 1" and "Set 2" in Table II, we were interested to evaluate the quality of the result obtained with TM as the size of the system increases. Thus, we have used 7 synthetic benchmarks, with the number of network nodes ranging between 16 and 55 nodes. The first set of 4 benchmarks have a load of 50%, and the second set of benchmark have a load of 70%. As we can see, TM is able to signigicantly improve the cost function over SSM, even as the size of the system increases. We used a time limit of 45 minutes for the first set and 90 minutes for the second set.

Finally, we used one real-life benchmark derived from [11], based on the SAE automotive communication benchmark [1]. In this benchmark we have 18 network nodes (ESes and NSes), and 83 frames (with the parameters) generated based on the messages presented in [11]). Table II contained the results for this benchmark – the last line labeled with "Set 3". The results obtained for the real-life benchmark confirms the results of the synthetic benchmarks.

6 Conclusions

We have presented an approach to the analysis and optimization of mixed-criticality applications on partitioned architectures. Applications of different criticality levels can be integrated onto the same architecture only if there is enough spatial and temporal separation among them. We have considered IMA for the PE-level Separation and TTEthernet for the communication-level. As the experimental evaluations shows, only by optimizing the implementation of the applications, taking into account the particularities of IMA and TTEthernet, we are able to support the designer in obtaining schedulable implementations.

Acknowledgements

This work has been funded by the Advanced Research & Technology for Embedded Intelligence and Systems (ARTEMIS) within the project 'RECOMP', support code 01IS10001A, agreement no. 100202.

References

[1] SAE Technical Report J2056/1. Technical report, SAE International.
[2] *AS6802: Time-Triggered Ethernet*. SAE International, 2011.
[3] N. Audsley and A. Wellings. Analysing APEX applications. In *Real-Time Systems Symp.*, pages 39 –44, 1996.
[4] S. Baruah and G. Fohler. Certification-Cognizant Time-Triggered Scheduling of Mixed-Criticality Systems. *Proceedings of the Real-Time Systems Symposium*, pages 3–12, 2011.
[5] G. C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, 1997.
[6] R. Dick. Embedded system synthesis benchmarks suite. http://ziyang.eecs.umich.edu/dickrp/e3s/.
[7] R. Ernst. Certification of trusted mpsoc platforms. 10th International Forum on Embedded MPSoC and Multicore, 2010.
[8] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.
[9] B. Leiner, M. Schlager, R. Obermaisser, and B. Huber. A Comparison of Partitioning Operating Systems for Integrated Systems. *Computer Safety, Reliability, and Security*, pages 342–355, 2007.
[10] S. O. Marinescu, D. Tămaş-Selicean, V. Acretoaie, and P. Pop. Timing Analysis of Mixed-Criticality Hard Real-Time Applications Implemented on Distributed Partitioned Architectures. 2012.
[11] U. Mohammad, N. Al-holou, and P. D. Development of an automotive communication benchmark. *Canadian Journal on Electrical and Electronics Engineering*, 1(5):99–115, 2010.
[12] J. Rushby. Partitioning for avionics architectures: Requirements, mechanisms, and assurance. NASA Contractor Report CR-1999-209347, NASA Langley Research Center, June 1999.
[13] W. Steiner. Synthesis of Static Communication Schedules for Mixed-Criticality Systems. In *Proceedings of the International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, pages 11 –18, 2011.
[14] D. Tămaş-Selicean and P. Pop. Design Optimization of Mixed-Criticality Real-Time Applications on Cost-Constrained Partitioned Architectures. In *Proceedings of the Real-Time Systems Symposium*, pages 24–33, 2011.