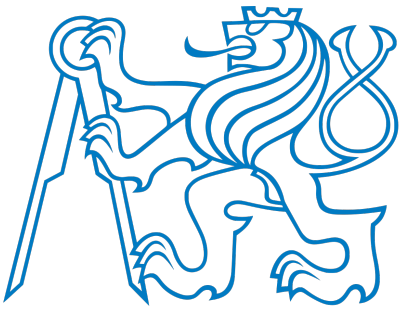

TECHNICAL REPORT

Time-Triggered Co-Scheduling of Computation and Communication with Jitter Requirements

Authors:
Anna MINAEVA, Dakshina DASARI, Zdeněk HANZÁLEK and Benny AKESSON

1 Introduction

This document is an internal technical documentation to the program CC_Scheduling_WithJitter which implements the solution described in article [1] and to the benchmarks used in the article. It allows to solve the problem of mapping and scheduling safety-critical functionality of time-triggered embedded systems with constrained jitter requirements to Engine Control Units (ECU) and on-chip interconnect in the automotive domain. A user can choose to have an optimal solution using either Integer Linear Programming (ILP) or Satisfiability Modulo Theory (SMT) technique for problems with up to 80 activities (either tasks running on cores or messages transmitting over interconnects). Moreover, a heuristic solution can be obtained by the heuristic approach 3-LS heuristic that can solve problem instances with more than 10000 activities. Both optimal and heuristic approaches use IBM ILOG CPLEX Optimization Studio v. 12.5.1 and Z3 theorem prover v. 4.5.0.

The benchmarks are obtained from collaboration with Bosch ([2]). Five sets each with 100 synthetic instances of different sizes are presented to validate the proposed approach. Moreover, the case-study representing an Engine Management System (EMS) that is responsible for controlling the time and amount of air and fuel injected by the engine by considering the values read by numerous sensors in the car, is presented.

2 Project Configuration

The project is written in Java programming language with Java Development Kit 8. The entry point of the project is 'CC_Scheduling_WithJitter.java' that contains main function and the implementation of all experimental environment. To run the project, a user should install IBM ILOG CPLEX Optimization Studio library and add 'cplex.jar' to the project. Besides, the Z3 theorem prover must be installed and added to the project (âĂIJcom.microsoft.z3.jarâĂİ). Moreover, 'Project Properties→Run→VM" field should contain '-Djava.library.path="/path_to_ibm/IBM/ILOG/CPLEX_Studio126 /cplex/bin/x86-64_osx":"path_to_z3/z3/build"' or similar.

3 Project Structure

There is one package in the problem that is called SC and it contains 17 classes, described below:

1. *CC_Scheduling_WithJitter* is the entry point to the project that contains all the experiment settings and parameters, described in the following section.

2. *FullSMT* contains SMT model implementation.
3. *ILP_model* is a parent class, containing necessary methods for ILP.
4. *FullILP* is a class for the ILP model with computation time improvements from Section 4.1 of [1].
5. *SubModel* is a class for the sub-model used in 3-LS heuristic.
6. *ThreeLSheuristic* that implements 3-LS heuristic, described in Section 5 of [1].
7. *ScheduledActs* contains a set of already scheduled activities, used by the 3-LS heuristic.
8. *RunnablesToECUsMapper* implements the ILP model that does mapping of tasks to cores.
9. *SolutionMapping* is a class with a mapping problem solution.
10. *SolutionOneAct* is a class that contains scheduling problem solution for one activity, used by the heuristic.
11. *Solution* contains the scheduling problem solution with all activities scheduled at once.
12. *ProblemInstanceSingleChain* is necessary to do the mapping, containing distinct data dependency chains.
13. *ProbInstSched* is a class for problem instance with mapping. All of the proposed approaches work with instances of this class.
14. *ProblemInstanceMapping* is a problem instance class before the mapping is done.
15. *DAG* is implemented for a directed acyclic graph of data dependencies.
16. *Activity* is a class for one distinct activity.
17. *Helpers* is a class with auxiliary functions and variables, not related directly to any of the existing classes.

4 Tunable Parameters and Experiments

In the method *main()* of *CC_Scheduling_WithJitter* class different experiments can be run. First of all, the experiment with different percents of zero-jitter activities from Section 6.2.4 can be run by setting parameter *IS_EXPERIMENT_WITH_DIF_PERCENTS_OF_ZJ_ACTS* to value *true*. Secondly, by setting parameter *IS_EXPERIMENT_WITH_DIF_PERIODS* to *true* one can activate the experiment with different percent of non-harmonic periods from Section 6.2.5.

All experiments are done over instances from one of the 5 sets (folders instances/Set 1-Set 5) or on the use-case (folder instances/Case study), where the range of instances is controlled by parameters *MIN_NUMB_INSTANCES* and *MAX_NUMB_INSTANCES*. Then, cycle iterators *solve_optimally* and *solve_with_zero_jitter* determine whether one of the optimal approaches or the heuristic will be used and whether all jitter requirements will be set to 0 or not, respectively. Moreover, cycle iterator *opt_sol_method* defines which model will be used - either ILP or SMT. Finally, all experiments run by gradually increasing utilization of all resources, which is controlled by iterator *util_on_res*.

Other parameters include: *N_CORES_MAIN_EXP* which defines number of cores in the considered architecture; *IS_GIVEN_MAPPING* that indicates whether the mapping is given from scratch (which is not given in Bosch problem instances); *TIME_LIMIT* time limit for the exact approaches; *IS_JITTER_RELATIVE* that allows to set jitter either relative or absolute, the definition can be found in [1]; *IS_USE_CASE* indicates that it is a case study, where the utilization is not scaled; *SOLVE_SUBMODEL_BY_ILP* sets the submodel to be solved by ILP or SMT; *IS_VERBOSE* sets verbosity; *COEFFICIENT_FOR_JITTER* sets the coefficient for jitter-constrained activities; and *UNSCHED_RULE* defines the rule to unschedule activities in the heuristic.

5 Benchmarks

The benchmarks, generated by Bosch tool [2] have the following parameters:

- *NumChains* defining number of cause-effect chains;
- *numberOfTasksinChain* is the number of tasks in each chain;
- *ProcessingTimesOfMessages* defines processing times of messages;
- *processingTimesRunnables* is processing times of tasks;
- *periods* defines periods of tasks;
- *communicationTimeForOrderCriticalMessages* is communication times of order critical messages;
- *sizeOfRunnables* gives size of tasks;
- *SenderLabelReceiverOrderCriticalChains* and *SenderLabelReceiverNonOrderCriticalChains* define a sender that send labels, labels that are transmitted and the receiver that gets the label for order-critical and non-order-critical chains;

6 Experiments

This section experimentally evaluates and compares the proposed optimal models and 3-LS heuristic on synthetic problems with jitter requirements set differently to show the benefits of JC scheduling in terms of utilization. Furthermore, we quantify the trade-off of additional cost in terms of memory to store the schedule and increase in computation time versus this gained utilization. Note that the goal of this section is to show the advantages and disadvantages of the JC approach. The experimental setup is presented first, followed by experiments that evaluate the proposed exact and heuristic approaches for different jitter and period requirements. We conclude by demonstrating our approach on a case study of an Engine Management System with more than 10000 activities to be scheduled.

6.1 Experimental Setup

Experiments are performed on problem instances that are generated by a tool developed by Bosch [2]. There are five sets of 100 problem instances, each set containing 20, 30, 50, 100 and 500 tasks, respectively. The same problem instance is presented with different jitter requirements. The generation parameters for each dataset are presented in Table 1, and the granularity of the timer is set to be 1 μs. Message communication times are computed for the considered platform with the following parameters: bandwidth $bnd = 400$ MB/s and latency $lat = 50$ clock cycles.

The mapping is found by *RunnablesToECUsMapper* so that load is balanced across the cores, i.e., the resulting mapping utilizes all cores approximately equally. The resulting problem instances contain 30-45, 50-65, 90-130, 180-250 and 1500-2000 activities (tasks and messages) for sets with 20, 30, 50, 100 and 500 tasks, respectively.

While we initially assume a system with 3 cores connected over a crossbar (resulting in 6 resources), inspired by the Infineon Aurix Tricore Family TC27xT, the approach can scale to a higher number of cores, as shown in Section 6.2.4.

The metric for the experiments on the synthetic datasets is the maximum utilization for which the problem instance is still schedulable. The utilization is defined as $r_y = \sum_{a_i \in A: map_i = y} \frac{e_i}{p_i}$ on each resource $y = 1, \cdots, 6$. To achieve the desired utilization of each resource, the execution times of activities are scaled appropriately. The experiments always start from a utilization of 10%, increasing in steps of 1%, solving until the approach is not able to find a feasible solution. The last utilization value for which the solution was found is set as the *maximum utilization* of the approach on the problem instance. Although this approach to set the maximum schedulable utilization may not be completely fair, the utilization is monotonic in most cases. Therefore, we have chosen to approximate the results by setting this rule to get results that are easier to interpret.

Table 1: Generator parameters for the sets of problem instances

Set	$\|T\|$	P *[ms]*	*Variable accesses per task*	*Chains per task*
1	20	$1, 2, 5, 10$	4	4
2	30	$1, 2, 5, 10$	4	6
3	50	$1, 2, 5, 10, 20, 50, 100$	4	8
4	100	$1, 2, 5, 10, 20, 50, 100$	4	15
5	500	$1, 2, 5, 10, 20, 50, 100$	8	50

Experiments were executed on a local machine equipped with Intel Core i7 (1.7 GHz) and 8 GB memory. The ILP model and ILP part of the 3-LS heuristic were implemented in IBM ILOG CPLEX Optimization Studio 12.5.1 and solved with the CPLEX solver using concert technology, while the SMT model was implemented in Z3 4.5.0. The ILP, SMT and heuristic approaches were implemented in JAVA programming language.

6.2 Results

First, the experiments compare the computation time of the two optimal ILP and SMT approaches to show for which problem instances it is advantageous to use each of the approaches. Secondly, we evaluate trade-off between the maximum achievable utilization and computation time of the 3-LS heuristic and the optimal approaches for differently relaxed jitter requirements. Thirdly, since memory consumption to store the final schedule is also a concern, the trade-off between solution quality and required memory is evaluated for systems of different sizes. Finally, a comparison of different period settings is presented to show the applicability of the approach to different application domains and to evaluate the behavior of both ZJ and JC approaches for periods set differently. A time limit of 3 000 seconds per problem instance was set for the optimal approaches to obtain the results in reasonable time. Note that the best solution found so far is used if the time limit is hit.

6.2.1 Comparison of the ILP and SMT models with different jitter requirements

First of all, we compare the computation time distribution for Set 1 and Set 2 (of smaller instance sizes with 30-45 activities and 50-65 activities, respectively) for SMT and ILP approaches with jitter requirements of each activity $a_i \in A$ set to $jit_i = \frac{p_i}{2}$, $jit_i = \frac{p_i}{5}$, $jit_i = \frac{p_i}{10}$ and $jit_i = 0$. Since the first problem instance from Set 3 was computing for two days before it was stopped with no optimal solution found for both SMT and ILP models, the experiments with optimal approaches

Table 2: Number of instances optimal approaches failed to solve in 3 000 seconds

jit_i	$p_i/2$		$p_i/5$		$p_i/10$		0	
	Set 1	Set 2	Set1	Set 2	Set 1	Set 2	Set 1	Set 2
ILP	14	76	9	53	6	45	4	27
SMT	2	51	3	13	2	9	2	7

only use the first two sets. We will return to the larger sets in Section 6.2.3 when evaluating the 3-LS heuristic.

The distribution is shown in the form of box plots [3], where the quartile, median and three quartiles together with outliers (plus signs) are shown. Outliers are numbers that lie outside $1.5\times$the interquartile range away from the top or bottom of the box that are represented by the top and the bottom whiskers, respectively. Note that outliers were also successfully solved within the time limit.

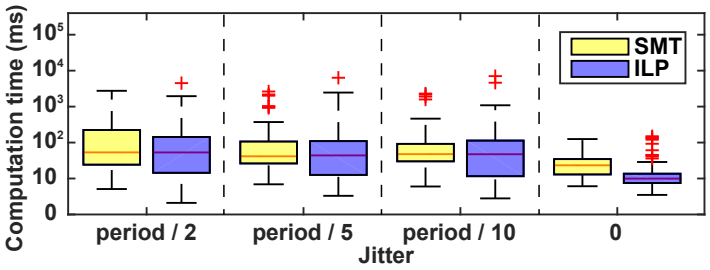

Figure 1: Computation time distribution for the SMT and ILP models with different jitter requirements for Set 1.

The number of problem instances from Set 1 and Set 2, that the optimal approaches failed to solve within the given time limit is shown in Table 2. Moreover, Figure 1 displays the computation time distribution on Set 1, where only problem instances, that both the ILP and SMT solvers were able to optimally solve all jitter requirements within the timeout period are included. For Set 1, it is 82 (out of 100), and for Set 2, it is 21 (out of 100) problem instances. The computation time distribution for Set 2 in show a similar trend, but since the sample is too small to be representative, we do not display them.

The results in Table 2 show that for more difficult problem instances the SMT model is significantly better than the ILP model in terms of computation time, since it is able to solve more problem instances within the given time limit. On the other hand, the comparison on the problem instances that both approaches were able to solve in Figures 1 indicates that the ILP runs faster on simpler problem instances that can be found at the bottom of the boxplots in Figure 6.2.1. As one can see, more relaxed jitter requirements result in longer computation time, which is a logical consequence of having larger solution space.

Thus, *the SMT model is more efficient than the ILP model for the considered problem on more difficult problem instances, while the ILP model shows better results for simpler instances, which justifies the usage of the ILP model in the 3-LS heuristic. Besides, more relaxed jitter requirements cause longer computation time for the optimal approaches*. Therefore, the SMT approach results are used for further comparison with the 3-LS heuristic.

6.2.2 Comparison of the optimal and heuristic solutions with different jitter requirements

Figure 2 shows the distribution of the maximum utilization of Set 1 for SMT and 3-LS heuristic with different jitter requirements. For comparison, we use the solution with the highest utilization, while the low value of initial utilization guarantees that at least some solution is found. The time limit caused 3 problem instances in Set 1 not to finish when using the SMT approach and these instances are not included in the results. The results for Set 2 are similar to that of Set 1, but due to small number of solvable instances we do not show them. The results

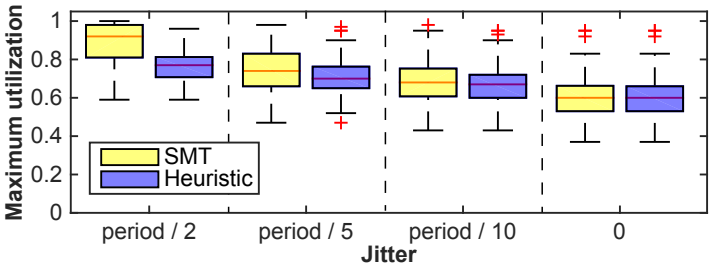

Figure 2: Maximum utilization distribution for the optimal SMT and 3-LS heuristic approaches with different jitter requirements for Set 1.

for the optimal approach show that stricter jitter requirements cause lower maximum achievable utilization. Namely, the average maximum utilization is 89%, 75%, 69%, 61% for Set 1 and 95%, 81%, 74%, 67% for Set 2 for the instances with jitter requirements equal to half, fifth, tenth of a period and zero, respectively. Meanwhile, the comparison of the 3-LS heuristic to the optimal solution reveals that the average difference goes from 17% and 23% for Set 1 and Set 2, respectively, with the most relaxed $jit_i = \frac{p_i}{2}$ to 0.1% for both sets with ZJ scheduling. This difference for problem instances with more relaxed jitter requirements is caused by very large complexity of the problem solved. Furthermore, while the heuristic solves all problem instances in hundreds of milliseconds, the SMT model fails on 62 problem instances out of 200 within a time limit of 3 000 seconds. This reduction of the computation time by the heuristic is particularly important during design-space exploration, where many different mappings or platform instances

have to be considered. In that case, it is not possible to spend too much time per solution.

Hence, we conclude that *heuristic performs better with tighter jitter requirements and hence particularly well for ZJ scheduling, resulting in an average degradation of 7% for all instances. Moreover, unlike the SMT model, the 3-LS heuristic always finds feasible solutions in hundreds of milliseconds, hence providing a reasonable trade-off between computation time and solution quality.*

6.2.3 Comparison of the heuristic with ZJ and JC scheduling

While the previous experiment focused on comparing the optimal approach and the heuristic, therefore using only smaller problem instances, this experiment evaluates the 3-LS heuristic on all sets. Due to time restrictions, only two jitter requirements were considered, $jit_i = \frac{p_i}{5}$ and $jit_i = 0$. Figure 3 shows the distribution of the maximum utilization for the 3-LS heuristic on Sets 1 to 5. In all sets, 100 problem instances were used for this graph. The results show that with growing size of the problem instance, the maximum utilization generally increases. The average difference in maximum utilization of the 3-LS heuristic on the problem instances with JC and ZJ requirements is 15.3%, 9.7%, 8.6%, 4.2% and 7.5% for Sets 1 to 5, respectively, with JC achieving higher utilization. The decreasing difference with growing sizes of the problem is caused by the growing average utilization. For instance, the average maximum utilization for Set 5 is 89.1% for the problem instances with JC requirements and 82.6% for the problem instances with ZJ requirements, pushing how far the maximum utilization for the JC scheduling can go. This tendency of increasing maximum utilization for the ZJ scheduling can be intuitively supported by the fact that more and more activities are harmonic with each other, which results in easier scheduling. In reality, harmonization costs a significant amount of over-utilization, especially when activities with smaller periods are concerned. On problem instances without harmonized activity periods the JC scheduling can show notably better results for larger instances compared to ZJ scheduling, as shown in Section 6.2.5.

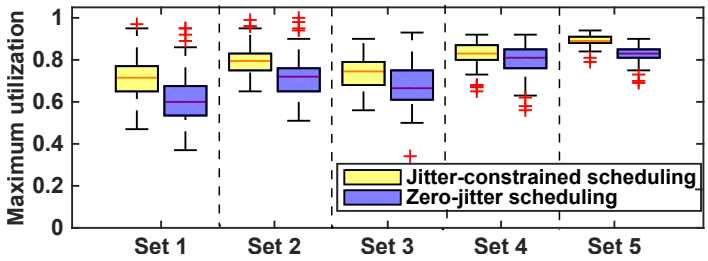

Figure 3: Maximum utilization distribution for the 3-LS heuristic with jitter-constrained and zero-jitter requirements in sets with 20, 30, 50, 100 and 500 tasks.

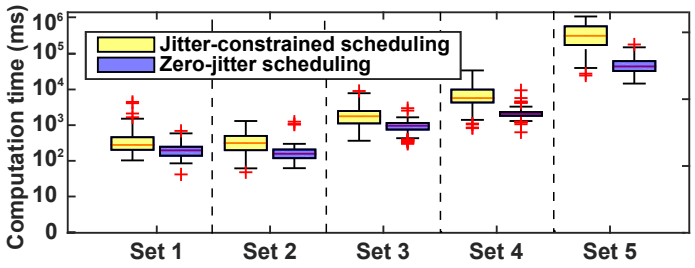

Figure 4: Computation time distribution for the 3-LS heuristic with jitter-constrained and zero-jitter requirements in sets with 20, 30, 50, 100 and 500 tasks.

Figure 4 shows the computation time of ZJ and JC using the 3-LS heuristic. Similarly to the optimal approach, the 3-LS heuristic takes longer to solve problem instances with JC requirements due to the larger solution space. Specifically, the average computation time for JC heuristic for Sets 1 to 5 are 0.3, 0.6, 3.6, 14.5 and 1003.6 seconds, respectively, while for ZJ scheduling it is 0.15, 0.28, 1.6, 4 and 109 seconds. Thus, solving a problem instance with JC requirements with 1500-2000 activities takes less than 17 minutes on average, which is still reasonable. Hence, *the 3-LS heuristic with JC scheduling provides better results, but requires more time than the 3-LS heuristic with ZJ scheduling*.

To summarize this experiment, JC scheduling *is promising in terms of maximum utilization, as it schedules with up to 55% higher resource utilization*. Besides, the computation time of the proposed heuristic is affordable even for larger problem instances, while the optimal models fail to finish in reasonable time already for much smaller instances. Moreover, *the proposed heuristic solves the problem instances with ZJ requirements near-optimally with a difference of 0.1% in schedulable utilization on average*. Generally, *the JC heuristic provides more efficient solutions than the ZJ heuristic, while requiring longer computation time*.

6.2.4 Evaluation of required memory and maximum utilization trade-off with different number of cores

The trade-off between maximum achievable utilization and the amount of memory required to store the schedule is evaluated by this experiment. Figure 5 shows the average maximum utilization achieved on systems with different number of cores and with gradually increasing percentage of JC jobs on 50 problem instances from Set 2 (due to time restrictions). The jitter constraint is set to $jit_i = \frac{p_i}{5}$ and the instances are solved to optimality. Furthermore, the problem instances with different numbers of cores are solved in steps of 5% of jobs with zero-jitter requirement, which reflects how much memory is necessary to store the schedule for such solutions. Note that the execution times of the activities are scaled proportionally to the number of cores so that each resource has a required utilization.

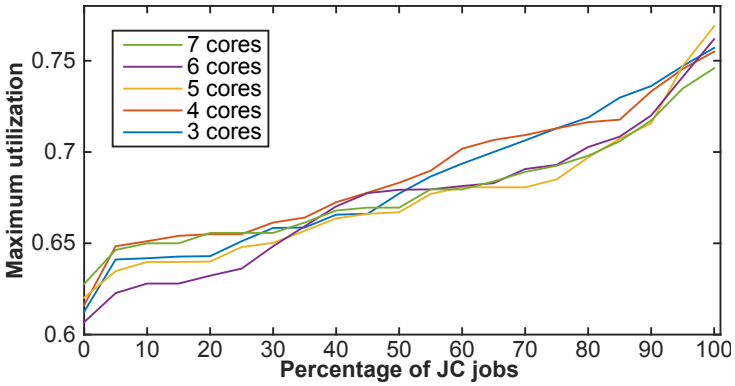

Figure 5: Utilization distribution for different percents of JC activities for different architectures.

The results show that introducing more JC jobs and thus increasing memory requirements for storing the final schedules can significantly improve the average maximum utilization. Namely, for the architecture with 4 cores, the maximum utilization with all ZJ jobs is 61%, while relaxing jitter requirements of half the jobs results in 69% utilized resources, and relaxation all of the jobs increases the maximum utilization to 76%. Concerning the required memory to store the schedule, the problem instances with 4 cores on average contain 80 jobs with JC scheduling and 49 jobs while scheduling in zero-jitter manner. Thus, assuming we need 8 bytes to store the schedule of one job, the memory overhead of relaxing jitter is 31 * 8 = 248 bytes, which is a reasonable price to pay for utilization gain of 15% on average on each resource.

Concerning the increasing number of cores, the results demonstrate that on average there is no significant dependency on how much cores we have in the system. Hence, *JC scheduling can result in high utilization gain, although at the cost of increased memory requirements to store the resulting schedule*.

6.2.5 Comparison of the different period settings

To show that the approach is applicable to other domains, an experiment with different period settings is performed. All problem instances from Set 2 are solved monoperiodically ($p_i = 10$ ms for each activity $a_i \in A$), or with harmonic periods (activities with $p_i = 2$ ms are changed to $p_i = 5$ ms), or with initial periods (i.e. with periods 1, 2, 5, 10 ms), or with non-harmonic periods (with periods 2, 5, 7, 12 ms). Figure 6 displays the average maximum utilization achieved by the 3-LS heuristic with ZJ and JC scheduling ($jit_i = \frac{p_i}{5}$) on 100 problem instances from Set 2. Since the optimal approach was not able to solve 7 out of 10 first instances with non-harmonic periods within the given time limit, due to its complexity and

extended hyper-period, the optimal approach results are not included in the figure.

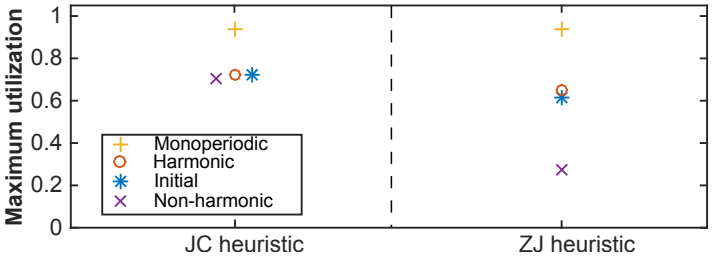

Figure 6: Utilization distribution for problem instances with different periods.

The results show that the maximum utilization for both ZJ and JC is achieved when scheduling monoperiodically, which is explained by having less possible collisions in the resulting schedule. An interesting observation is that for JC scheduling all other period settings on average resulted in very similar maximum utilization, while the ZJ approach shows the variation of 27% with non-harmonic periods, 62% with initial periods and 65% with harmonic period set. Relative insensitivity of JC scheduling to period variations can be caused by significantly larger solution space due to relaxation of strict jitter constraints. This allows to find solutions with high utilization even with the non-harmonic period setting.

Besides, the same order of computation time distribution is shown by different period settings, i.e. monoperiodic scheduling is the fastest, while the problem instances in the non-harmonic period set result in the longest computation time.

Thus, *the proposed approach is applicable to other domains, where the application periods have different degree of harmonicity. Furthermore, increasing harmonicity of the period set results in higher maximum utilization, lower computation time and lower gain of JC scheduling in comparison with ZJ scheduling in terms of maximum utilization*.

6.3 Engine Management System Case Study

We demonstrate the applicability of the proposed 3-LS heuristic on an Engine Management System (EMS). This system is responsible for controlling the time and amount of air and fuel injected by the engine by considering the values read by numerous sensors in the car (throttle position, mass air flow, temperature, crankshaft position, etc). By design, it is one of the most sophisticated engine control units in a car consisting of 1000-2000 tightly coupled tasks that interact over 20000 to 30000 variables, depending on the features in that particular variant. A detailed characterization of such an application is presented by Bosch in [2], along with a problem instance generator that creates input EMS models in conformance with the characterization.

We consider such a generated EMS problem instance, comprising 2000 tasks with periods 1, 2, 5, 10, 20, 50, 100, 200 and 1000 ms and with 30000 variables in total, where each task accesses up to 12 variables. There are 60 cause-effect chains in the problem instance with up to 11 tasks in each chain. We consider the target platform to be similar to an Infineon AURIX Family TC27xT with a processor frequency of 125 MHz and an on-chip crossbar switch with a 16 bit data bus running at 200 MHz, thus having a bandwidth of 16-bit x 200 MHz / 8 = 400 MB/s. The time granularity is 1 μs, and the resulting hyper-period is 1000 ms. However, setting the hyper-period to be 100 ms results in a utilization loss of less than 0.5%, arising from shortening the scheduling periods of tasks with periods 200 ms and 1000 ms and over-sampling, which is a reasonable sacrifice to decrease the memory requirements of the schedule. The tool in [2] provides the number of instructions necessary to execute each task, which is used to compute the worst-case execution time with the assumption that each instruction takes 3 clock cycles on average (including memory accesses that hit/miss in local caches).

The mapping of tasks to cores requires minimally 3 cores with the utilization approximately 89.6% on each core and approximately 30% on each input port of the crossbar. Moreover, the resulting scheduling problem has 10614 activities with 104721 jobs for the JC assumptions in total. Neither SMT nor ILP can solve this problem in reasonable time, but the JC heuristic with $jit_i = \frac{p_i}{5}$ for all a_i solves the problem in 43 minutes. By gradually introducing more activities a_i with $jit_i = 0$, we have found a maximum value of 85% ZJ activities for which the 3-LS heuristic is still able to find a solution, which takes approximately 12 hours. Note that the computation time has increased with introducing more ZJ activities due to more restricted solution space. However, to store the schedule in the memory for 0% ZJ jobs, 104721 * 8 = 818 Kbytes of memory is required assuming that one job start time needs 8 bytes, while with 85% ZJ jobs it is only 19394 * 8 = 152 Kbytes. *Thus, for realistic applications the optimal approaches take too long, while the 3-LS heuristic approach is able to solve the problem in reasonable time. Moreover, increasing the percent of ZJ activities has shown to provide a trade-off between computation time and required memory to store the obtained schedule.*

References

[1] A. Minaeva, D. Dasari, Z. Hanzalek, and B. Akesson, "Time-triggered co-scheduling of computation and communication with jitter requirements," *submitted to ESWEEK 2017*.

[2] S. Kramer, D. Ziegenbein, and A. Hamann, "Real world automotive benchmark for free," *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-Time Systems*, 2015.

[3] P. Kampstra *et al.*, "Beanplot: A boxplot alternative for visual comparison of distributions," *Journal of statistical software*, vol. 28, no. 1, pp. 1–9, 2008.