저작자표시-비영리-변경금지 2.0 대한민국

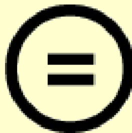
이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

공학석사 학위논문

Functionally and Temporally Correct Simulation for Automotive Systems on Multicore Simulator

멀티코어 시뮬레이터에서의 자동차 시스템을 위한 기능적/시간적 정확성 보장 시뮬레이션 기법

2019년 2월

서울대학교 대학원
컴퓨터공학부
이 원 석

Functionally and Temporally Correct Simulation for Automotive Systems on Multicore Simulator

지도교수 이 창 건

이 논문을 공학석사 학위논문으로 제출함

2018년 11월

서울대학교 대학원

컴퓨터공학부

이 원 석

이 원 석의 석사 학위논문을 인준함

2018년 12월

위 원 장	하 순 회	(인)
부위원장	이 창 건	(인)
위 원	이광근	(인)

Abstract

Functionally and Temporally Correct Simulation for Automotive Systems on Multicore Simulator

Wonseok Lee
Department of Computer Science and Engineering
The Graduate School
Seoul National University

This dissertation presents functionally and temporally correct simulation method for cyber-side of an automotive system on multicore simulator. To overcome the limitations of the existing simulation methods which do not correctly model temporal behaviours such as varying execution times and task preemptions, the novel simulation technique assuming single core simulator was proposed. In this work, we extend the single core simulator to the multicore while keeping all of the proposed key ideas to guarantee correct simulation. We introduce heuristic task partitioning algorithm based on memory usages and approximated task-wise blockings of simulated tasks. As a result, we could improve up to 97%p, 15%p of simulation capacity compared to the single core, and other task partitioning algorithms, respectively.

keywords : Automotive System Simulation, Real-time Simulation

Student Number : 2017-21586

Contents

1 Introduction **1**

1.1 Motivation and Objective 1

1.2 Approach 1

1.3 Organization 2

2 Related Work **3**

2.1 Model-Based Simulations 3

2.2 Real-time Execution Platforms 3

2.3 Functionally and Temporally Correct Simulations 3

3 Background **5**

3.1 Description on the real cyber-system 5

3.2 Description on the simulated cyber-system 7

3.3 Idea of Functionally and Temporally Correct Simulation 9

4 Problem Description **12**

4.1 Keeping the key ideas of the single core simulator 12

4.2 Maximally utilizing the multicore 13

5 Proposed Approach **15**

5.1 Memory constraint 15

5.2 The Smallest-blocking-first heuristic 18

5.2.1 Intuition of Smallest-blocking-first algorithm 19

5.2.2 Finding the Expected Earliest Start Time 20

5.2.3 Finding the Expected Latest Finish Time 22

5.2.4 Weighting the $[EEST_{ij}, ELFT_{ij}]$ intervals 25

6 Evaluation **28**

6.1 Simulatability according to the number of cores 28

6.2 Simulatability according to the partitioning method 30

6.3 Simulatability according to the physical read/write task ratio 31

7 Conclusion **35**

References **37**

List of Figures

1 Predicted performance and real performance of LKAS 2

2 Example automotive system 6

3 Execution scenario and simulation scenario of example automotive system 6

4 Job-level precedence graph of the example automotive system . . . 10

5 Possible job-level precedence graphs after executing J_{11} on the simulator 14

6 Memory size of functions for body control module implemented by Renault 16

7 Intuition of Smallest-blocking-first heuristic 19

8 Construction of sparse graph and $EEST_{ij}$ of each job for the example job-level precedence graph 21

9 Construction of dense graph and $ELFT_{ij}$ of each job for the example job-level precedence graph 23

10 Weighted intervals and task-wise blocking values for the example job-level precedence graph 26

11 Simulatability according to the number of simulator cores 29

12 Simulatability compared to the other task partitioning heuristics . . 31

13 Simulatability according to physical read/write task ratio 32

List of Tables

1 Normalized worst case execution times of tasks according to co-run task 17

2 Normalized worst case execution times of tasks according to the memory usage of each core 17

1 Introduction

1.1 Motivation and Objective

Simulating the automotive system using an accurately designed simulation model is essential to correctly predict its final performance at design phase. Incorrect prediction brought out by imprecisely modeled simulation causes painful revalidation on following development processes. Figure 1 shows that Simulink [1] predicts the ideal performance of LKAS (Lane Keeping Assistance System) which does not match with its real performance. This real performance is only revealed at the end of the implementation phase, and subsequently system developers ought to repeat the entire development processes from the beginning.

The reason why the real performance is not predicted on the existing simulation tools is that they only focus on the functional behaviors of the system and do not carefully consider the temporal behaviors such as varying execution times and task preemptions on ECU (Electronic Control Unit) environments. To consider not only the functional behaviors but also the temporal behaviors of the automotive system, the previously proposed simulation method transformed the simulation problem to a real-time task scheduling problem on single core simulator PC [2].

1.2 Approach

In this paper, we extend the proposed single core simulator to the multicore simulator considering:

- **memory constraint** which is necessary to guarantee functionally and tempo-

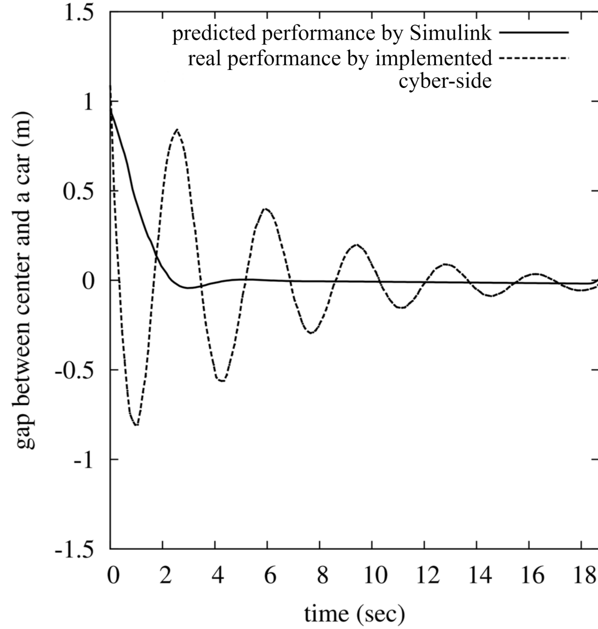


Figure 1: Predicted performance and real performance of LKAS [3]

rally correct simulation

- **heuristic task partitioning algorithm** to efficiently utilize the multicore by minimizing approximated task-wise blockings among the simulated tasks

1.3 Organization

This paper is organized as follows. In Section 2, we survey related works. Then, Section 3 explains the idea of functionally and temporally correct simulation on single core simulator that will be extended to the multicore. In Section 4, we describe our problem. In Section 5, we propose our approach. In Section 6, we evaluate our approach through synthetic workload experiments. Finally, Section 7 concludes the paper.

2 Related Work

2.1 Model-Based Simulations

To predict the final performance of cyber-side of an automotive system at design phase, simulation tool such as Simulink [1] is widely used in industry. However, it mimics only functional behaviours of the system and does not consider temporal behaviours which will occur once the system is implemented on the ECUs. The simulated tasks on Simulink are ideally executed while ignoring the temporal differences caused by ECU environments. Moreover, Simulink is focusing on offline simulations which do not interact with physical-side.

2.2 Real-time Execution Platforms

To simulate the system while interacting with the physical-side, real-time simulation on AutoBox [4] is commonly used. However, AutoBox provides only rapid prototyping of the system and does not consider the real ECUs' performance. The temporal behaviours of the simulated tasks are determined only by the performance of AutoBox hardware, and users do not have any control knob to model the real target ECUs' performance which determines the actual temporal behaviours and the final performance.

2.3 Functionally and Temporally Correct Simulations

To accurately model both of the functional and temporal behaviours while interacting with the physical-side in real-time during the simulation the novel simulation method guaranteeing the functionally and temporally correct simulation was proposed [2].

However, the proposed method considered only single core simulator PC which does not provide enough capacity to simulate the whole system.

To increase capacity of the functionally and temporally correct simulation, a brief idea of multicore usage based on G-EDF (Global-Earliest Deadline First) scheduling was proposed [5]. However, in this approach, the task migration costs and memory interferences, which must be taken into to guarantee temporal correctness, were not considered.

3 Background

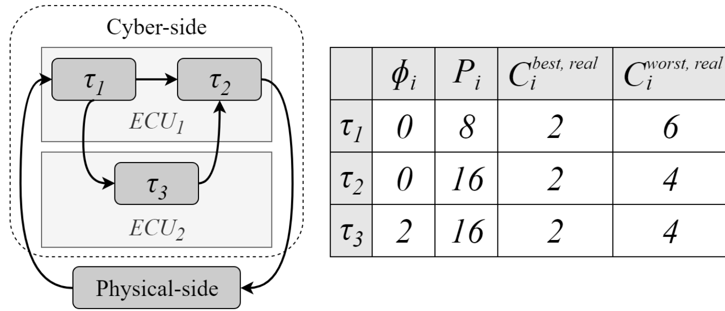
3.1 Description on the real cyber-system

The cyber-side of an automotive system to be simulated can be given as Figure 2 shows. Each control task is denoted as τ_i and data producer/consumer relations among the tasks or physical-side are denoted by directed edges as in Figure 2(a). Each task, τ_i, is realized on its mapped ECU as a periodic task and can be represented as a five-tuple:

$$\tau_i = (F_i, \Phi_i, P_i, C_i^{best,real}, C_i^{worst,real})$$

where F_i is the function that τ_i executes, Φ_i is the task offset, P_i is the period of τ_i. $C_i^{best,real}, C_i^{worst,real}$ represent the best/worst case execution time of τ_i on its mapped ECU, respectively.

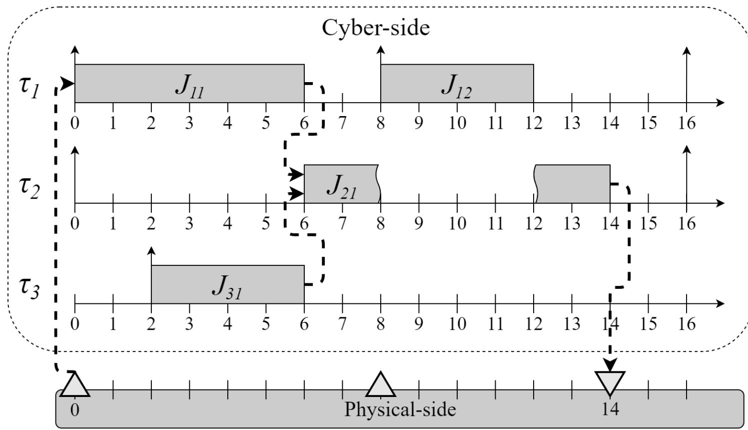
If the task parameters are given as Figure 2(b), and RM (Rate Monotinic) scheduling policy is assumed for each ECU, we can expect one of their possible execution scenarios as Figure 3(a) shows. Every j-th job of τ_i, J_{ij}, is released at $\Phi_i + (j-1)P_i$, and has execution time varying within $[C_i^{best,real}, C_i^{worst,real}]$. The time points where the cyber and physical-side interact each other are marked as triangles, and one of the data paths from the physical read to the physical write is denoted by dashed directed arrows. In the Figure 3(a), J_{11}, who is the job of the τ_1 that reads physical-side data, reads data from the physical-side at time 0 and produces output for its consumer task's job, J_{21}. J_{21}, who starts after its producer tasks' jobs J_{11} and J_{31} finish, consumes the outputs produced by J_{11} and J_{31}. After that, J_{21} writes its output to the physical-side at time 14. J_{31}, who starts before its producer task's job J_{11} finishes,



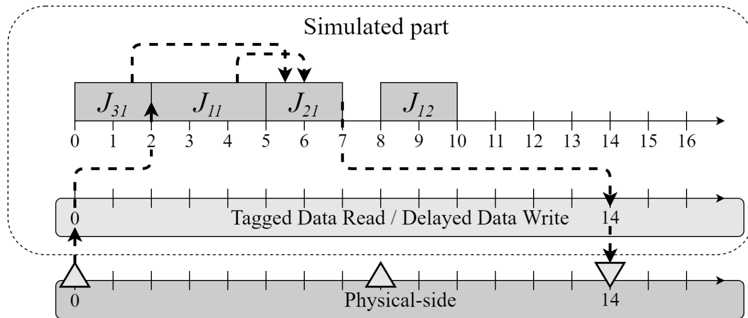
	ϕ_i	P_i	$C_i^{best, real}$	$C_i^{worst, real}$
τ_1	0	8	2	6
τ_2	0	16	2	4
τ_3	2	16	2	4

(a) Cyber-side of an automotive (b) Control task parameters

Figure 2: Example automotive system



(a) Possible execution scenario of example automotive system



(b) Simulation scenario of example automotive system

Figure 3: Execution scenario and simulation scenario of example automotive system

may consume the data produced by J_{10} (not shown in this figure).

At the above scenario, we assumed that all of the F_is always consume the most recently produced data at the entry of their executions and produce their output at the exit of the executions. On the same assumptions, let's additionally assume that the simulator running on a PC environment has:

- **Faster execution than ECU:** Since PC has more powerful performance than ECU, the execution times of F_is are much faster on the simulator than that of on the ECU. e.g., Core i7-9700K [6] in PC vs. TC275 [7] in ECU.
- **Tagged/Delayed Data Read/Write:** The simulator can log all of physical read/write data with time-tags. The simulator can execute the F_is with any specific tagged physical read data. Similarly, the simulator can write the delayed output data to the physical-side at any specific time point.
- **Execution time mapping functions:** For every F_i, there exist execution time mappings between the simulator and the ECU. That is, when J_{ij} is executed on the simulator for the time of e_{ij}^{sim}, we can estimate its execution time on the ECU, $e_{ij}^{real} = M_i(e_{ij}^{sim})$ where M_i represents the execution time mapping function. This assumption on multicore simulator will be validated in Section 5.

3.2 Description on the simulated cyber-system

On the above assumptions, the resulting physical-side interactions produced by the simulator depicted in Figure 3(b) equal with that of the real cyber-side shown in the Figure 3(a). In the Figure 3(b), we assumed that $e_{ij}^{real} = 2 * e_{ij}^{sim}$ for all F_is. At time 0, the simulator logs the physical-side data with its time-tag 0 while executing J_{31} who

may consumes the data produced by J_{10}. When J_{31}'s execution is finished, J_{11} starts its execution with the tagged physical-side data which was logged at time 0. After J_{11}'s execution is finished, J_{21} starts its execution. At this moment, its data producer jobs, J_{11} and J_{31}, just have finished. So, J_{21} consumes the produced data of J_{11} and J_{31} which is most recently produced. After J_{21}'s execution, J_{21} delays its physical write until 14 which is the same with the actual physical write time on the real cyber-side. In the Figure 3(b), we can see that the data path from the physical read to the physical write is the same with that of the real cyber-side in the Figure 3(a). The simulator shown in the Figure 3(b) guarantees the functional and temporal correctness since it executes the same F_is with the same inputs and receives/gives the same interaction from/to the physical-side at the same time point as the real cyber-side. More formally, we can say that the simulation is functionally and temporally correct if all of the simulated jobs can be scheduled while satisfying:

- **Physical-read constraint:** For any job J_{ij} who reads physical-side data, the simulator should schedule it later than its actual start time on the real cyber-side. i.e.,

$$t_{ij}^{S,sim} \geq t_{ij}^{S,real} \tag{1}$$

  where $t_{ij}^{S,sim}$ and $t_{ij}^{S,real}$ represent the start time of J_{ij} on the simulator and the real cyber-side, respectively.

- **Physical-write constraint:** For any job J_{ij} who writes its produced data to the physical-side, the simulator should finish it before its actual finish time on the real cyber-side, i.e.,

$$t_{ij}^{F,sim} \leq t_{ij}^{F,real} \tag{2}$$

where $t_{ij}^{F,sim}$ and $t_{ij}^{F,real}$ represent the finish time of J_{ij} on the simulator and the real cyber-side, respectively.

- **Producer/consumer constraint:** For any pair of jobs, $J_{i'j'}$ and J_{ij}, if $J_{i'j'}$ is a producer job of J_{ij} on the real cyber-side, the simulator should finish $J_{i'j'}$ before starting J_{ij}, i.e.,

$$t_{i'j'}^{F,sim} \leq t_{ij}^{S,real} \tag{3}$$

3.3 Idea of Functionally and Temporally Correct Simulation

To schedule the simulated jobs while meeting all of the above constraints, the simulator has to know $t_{ij}^{S,real}$ and $t_{ij}^{F,real}$ which are non-deterministic due to the varying execution times of the jobs. To tackle this challenge, the previously proposed simulation method [2] transformed the simulation problem to the scheduling problem of a job-level precedence. The proposed method progressively resolves such non-determinism by executing the simulated jobs during the simulation.

We skip the details of the proposed simulation method because they are beyond the scope of this paper. Instead, we briefly review the proposed method using the example automotive system in the Figure 2. The simulation problem for the cyber-side of an automotive system in the Figure 2 can be transformed to a job-level precedence graph scheduling problem as shown in Figure 4. At the left-side of the Figure 4, each vertex represents the job to be simulated. The tags, 'R' or 'W', at the upper-left corner of the jobs show the physical read/write constraints that the tagged jobs have. Each edge shows the pre-execution condition between the jobs where hat-job ($\hat{J}_{21}$) is virtually added job which has zero-execution time and is needed only for deriving pre-execution conditions. The solid edge $(J_{i'j'}, J_{ij})$ represents the deterministic

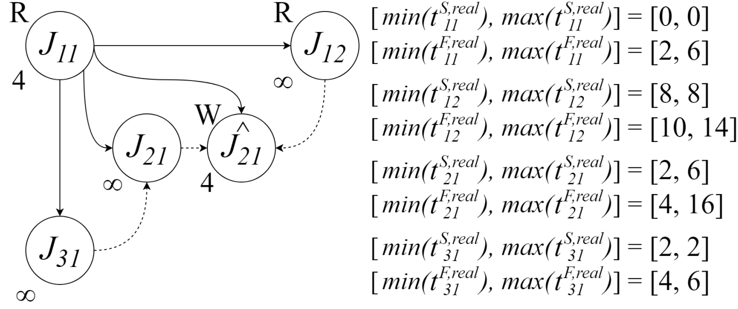


Figure 4: Job-level precedence graph of the example automotive system

edge which means $J_{i'j'}$ should be finished before J_{ij} starts. The dashed edge $(J_{i'j'}, J_{ij})$ represents the non-deterministic edge which means it is not known yet whether $J_{i'j'}$ should be finished before J_{ij} or not. The closed-intervals at the right-side of the Figure 4 represent the expected $t_{ij}^{S,real}$, $t_{ij}^{F,real}$ ranges which are varied by the execution times of the jobs. The numbers at the lower-left corner of the jobs show the deadlines which are calculated based on the deterministic edges and the $t_{ij}^{S,real}$, $t_{ij}^{F,real}$ ranges.

To schedule the jobs in the job-level precedence graph, the simulator first finds a job who does not have any unfinished deterministic predecessor. If the found job does not have the physical read constraint, it adds this job to the ready queue of the simulator. If the found job has the physical read constraint, it adds the found job only when Eq. 1 holds, i.e., current time is later than its start time on the real cyber-side. Out of the jobs in the ready queue, one of them is scheduled based on EDF (Earliest-Deadline-First) scheduling policy according to their assigned deadlines. Whenever a job in the ready queue is finished, its execution time on the simulator, e_{ij}^{sim}, becomes known, so its execution time on the real cyber-side, $e_{ij}^{real} = M_i(e_{ij}^{sim})$, is also known. Using the e_{ij}^{real}, the simulator progressively narrows the $t_{ij}^{S,real}$, $t_{ij}^{F,real}$ ranges. At this step, the non-deterministic edges are determined as deterministic edges or removed

based on the narrowed $t_{ij}^{S,real}$, $t_{ij}^{F,real}$ ranges. Lastly, the simulator re-assigns the deadline of each job using the updated job-level precedence graph. By iterating the above processes, the proposed simulation method can schedule the job-level precedence graph.

Meanwhile, it is already proven that at the time when the simulator is about to add a job to the ready queue, its start time on the real cyber-side, $t_{ij}^{S,real}$, is already known. Besides, it is also already proven that if the simulator can schedule all the simulated jobs meeting their assigned deadlines, they satisfy all of the constraints in Eq. 1, Eq. 2, and Eq. 3. i.e., functionally and temporally correct simulation. At following sections, we extends the above described single core simulator to the multicore.

4 Problem Description

We aim to extend the single core simulator to the multicore simulator while guaranteeing the functional and temporal correctness. The multicore simulator has multiple processing units (cores) and corresponding ready queues. It means that the multicore simulator can execute multiple simulated jobs in parallel. In this parallel execution environment, our goal is keeping the key ideas of the previously proposed single core simulator while maximally utilizing the benefits of parallel execution.

4.1 Keeping the key ideas of the single core simulator

The basis of the key ideas in the single core simulator is the execution time mapping function. If the execution time mapping still holds in the multicore simulator, the $t_{ij}^{S,real}$, $t_{ij}^{F,real}$ ranges can be correctly narrowed, and therefore the non-determinism in the job-level precedence graph can be properly resolved while keeping the temporal and functional correctness.

However, in the multicore environment, it was reported that the interferences between the cores at shared memory such as DRAM and shared cache cause the delay spike of the executed job as high as 600% of its normal execution time [8]. It means that e_{ij}^{sim} becomes unpredictable and is bound to the memory interferences, not to its actual computational amount. Therefore, e_{ij}^{real}, which is mapped from the e_{ij}^{sim} by the execution time mapping function, also becomes unpredictable. This unpredictability causes incorrect narrowing of the $t_{ij}^{S,real}$, $t_{ij}^{F,real}$ ranges and consequentially incurs incorrect simulation.

To prevent such unpredictability caused by the interferences between cores at

the shared memory, we introduce memory constraint at Section 5.1. By introducing the memory constraint, all the simulated jobs can be executed while minimizing the influence of the shared memory interferences between the cores.

4.2 Maximally utilizing the multicore

Our multicore simulator schedules the jobs in the job-level precedence graph. To schedule the jobs with precedence constraints on the multicore environment, G-EDF scheduling policy based on their effective deadlines is commonly used [9]. In this approach, effective deadline is assigned to each job along the fixed precedence relations between the jobs. By executing the jobs following the G-EDF scheduling policy according to their assigned deadlines, all the jobs with precedence constraints can be efficiently scheduled.

However, our job-level precedence graph scheduling problem has two major differences with the above problem. Firstly, our job-level precedence graph includes the non-deterministic edges, so its precedence relations between the jobs are not fixed and keep changing during the simulation. For example, both of (a) and (b) in Figure 5 are possible job-level precedence graphs after executing J_{11} in the example job-level precedence graph mentioned at the Figure 4. Depending on e_{11}^{real}, the non-deterministic edge, (J_{31}, J_{21}), becomes deterministic edge or is removed. Secondly, by introducing the memory constraint, we cannot use the G-EDF scheduling policy anymore. It is because that the G-EDF may cause a core to execute the tasks exceeding its memory usage limitation. The G-EDF scheduling policy globally picks a core to execute the tasks without any consideration about the memory usage.

To cope with such differences, we introduce heuristic task partitioning algorithm

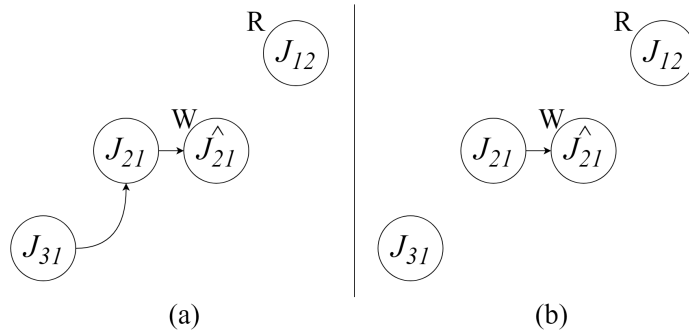


Figure 5: Possible job-level precedence graphs after executing J_{11} on the simulator

for partitioned EDF scheduling policy at Section 5.2. By using the heuristic task partitioning algorithm, the job-level precedence graph can be efficiently partitioned and scheduled on the multicore simulator.

5 Proposed Approach

As we mentioned before, we introduce the memory constraint for guaranteeing validity of execution time mapping functions. We limit the sum of tasks' memory usage on each core to minimize the influence of shared memory interferences between the cores. In other words, it means that the set of tasks, $\mathbb{T} = \{\tau_1, \tau_2, ...\}$, is partitioned into the each core according to the memory usage of each task, MEM_{τ_i}. Since there can be more than one partitionings satisfying the memory constraint, we have to find the most parallelizable partitioning among them to maximally utilize the benefits of the multicore. To also consider such chance to be parallelized, we introduce the heuristic task partitioning algorithm which minimizes the approximated task-wise blockings.

5.1 Memory constraint

To minimize the interfereces between the cores at the shared memory, the isolation techniques such as DRAM bank partitioning [10] and shared cache partitioning [11] were proposed. However, if we focus on the automotive system tasks which commonly have small memory usages, we can enjoy the reasonable level of isolation through the task partitioning alone without such precisely designed isolation techniques.

Since the automotive system tasks are run on ECU which has limited memory resource, they are normally implemented to access the small memory section. For example, Figure 6 shows the such restricted memory usages of the automotive functions composing the body control module of Renault (Due to the confidentiality reasons, the specific information of each function is not given). When we consider the parallel

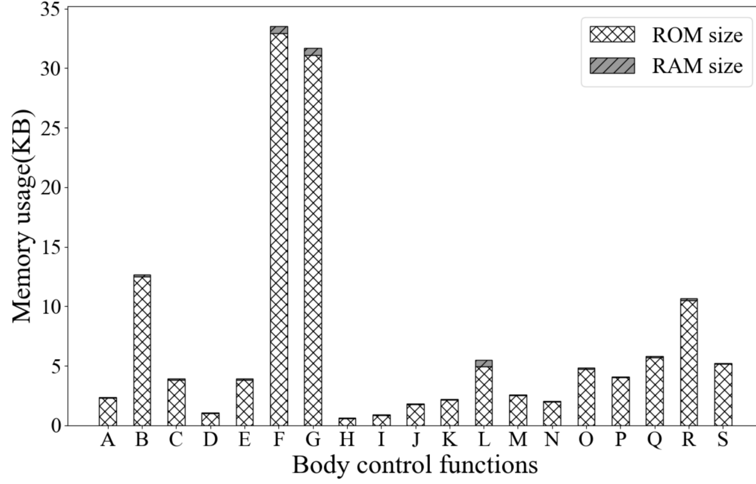


Figure 6: Memory size of functions for body control module implemented by Renault [12]

execution of such small memory usage tasks, the memory interferences on DRAM or shared cache are negligible since we can expect that they infrequently access to such shared memories. After the cold starts at the initial execution, their accessed memory blocks will be copied to the local cache of each core and rarely evicted since the other tasks on the same core also have small memory usages.

Table 1 shows the experiment result for clarifying the relation between the shared memory interferences and the amount of memory usages per each core. We parallelly executed two tasks at the different cores. Both of them access to float array in random order and calculate the sum of element-wise power. We varied their array size to 32KB, 4MB and executed them on i7-3610QM [13] which has 256KB local cache for each core. We measured their execution times during the 32 releases excluding the cold start. i.e., the first release. At every execution, the given array for each task remained the same. Each entry in the Table 1, $e_{ij}^{norm,max}$, represents the normalized worst execution time of τ_i which is measured on the simulator during the 31 releases.

Table 1: Normalized worst case execution times of tasks according to co-run task

Core1	Core2	$e_{1j}^{norm,max}$	$e_{2j}^{norm,max}$
τ_1:32k	τ_2:32k	1.0002	1.0025
τ_1:32k	τ_2:4m	1.0224	1.0652
τ_1:4m	τ_2:4m	1.0966	1.1050

Table 2: Normalized worst case execution times of tasks according to the memory usage of each core

Core1	Core2	$e_{1j}^{norm,max}$	$e_{3j}^{norm,max}$	$e_{5j}^{norm,max}$
τ_1:128k τ_2:128k τ_3:64k	τ_4:64k τ_5:32k τ_6:32k	1.1854	1.1880	1.0198
τ_1:128k τ_3:64k τ_5:32k	τ_2:128k τ_4:64k τ_6:32k	1.0054	1.0015	1.0020

They are normalized against the average execution time when they are executed exclusively, not in parallel. We can see that the noticeable delay spike, 10.5%, occurs only when the large memory usage task(4m)s run in parallel. For the small memory usage task(32k)s, the delays caused by parallel execution are negligible, up to 0.25%.

Although the interferences between the cores at the shared memory are negligible, limiting the amount of memory usage on the each core is still needed to make the execution times be more predictable. Table 2 shows such necessity of memory constraint. We executed 6 tasks which perform the same thing with the experiment in the Table 1 but in this time, tasks access to the array in sequential order and are given different periods. We varied their array size to 32KB, 64KB, 128KB and measured their execution times during the 32 releases excluding the first release. When we map the tasks to a core exceeding the local cache size of the core, the execution times of the tasks are delayed up to 18.8%. On the contrary, we can see that the delay spikes

do not exceed up to 0.5% when the sum of memory usages on each core does not exceed the local cache size of the core.

To limit the sum of memory usages for each core, we propose the task partitioning satisfying the following memory constraint:

$\forall c_i \in \mathbb{C}$,

$$\sum_{\forall \tau_i \; mapped \; to \; c_i} (MEM_{\tau_i}) \leq Local \; cache \; size \; of \; c_i \tag{4}$$

where $\mathbb{C} = \{c_1, c_2, ...\}$ represents the set of cores on the multicore simulator.

5.2 The Smallest-blocking-first heuristic

The problem finding the existence of the task partitioning which satisfies the memory constraint in Eq. 4 is reducible to bin packing decision problem which is known as NP-Complete [14]. When we consider exhaustive search of the whole possible partitioning cases, the size of solution space equals to $S(|\mathbb{T}|, |\mathbb{C}|)$ where S represents the second kind of Stirling number which exponentially increases according to the number of tasks and cores. e.g., $S(9,4) = 7770, S(10,4) = 34105$. Since there is no polynomial time algorithm and the whole solution space is too large to exhaustively search, the heuristic approaches such as Best-fit-first, Worst-fit-first can be considered to practically find the partitioning [15]. However, we cannot efficiently utilize the multicore through the above heuristics since they only focus on the packing of item(task) and do not consider the existence of the job-level precedence graph. Therefore, we propose Smallest-blocking-first heuristic which considers efficient parallelization of the job-level precedence graph.

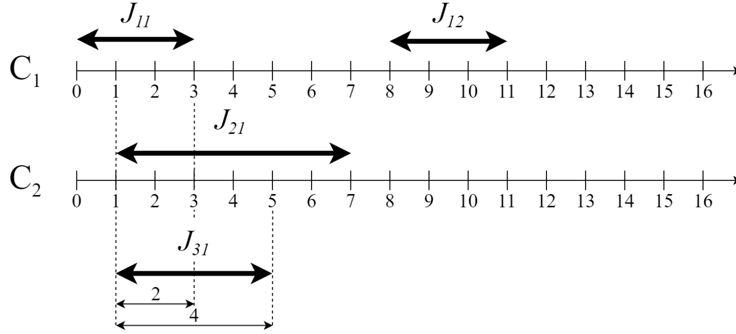


Figure 7: Intuition of Smallest-blocking-first heuristic

5.2.1 Intuition of Smallest-blocking-first algorithm

We explain the intuition of Smallest-blocking-first heuristic at Figure 7. Let's assume that the tasks τ_1 and τ_2 are already mapped to c_1 and c_2, respectively. Each bidirectional arrow represents the job of each task where its left-end means $EEST_{ij}$ (Expected Earliest Start Time) of J_{ij}, and its right-end means $ELFT_{ij}$ (Expected Latest Finish Time) of J_{ij}. When we determining which core the τ_3 is mapped to, we can calculate the interleaved $[EEST_{31}, ELFT_{31}]$ interval with the already mapped tasks. If the τ_3 is mapped to the c_1, the length of interleaved interval is 2 as Figure 7 shows. Since the only one job can be executed on a core at a time, one of the J_{11} and J_{31} will be blocked for the time of 2. Similarly, one of the J_{21} and J_{31} will be blocked for the time of 4 when the τ_3 is mapped to the c_2. The proposed Smallest-blocking-first heuristic always choose the smallest blocking core at every decision. In the example shown in the Figure 7, Smallest-blocking-first maps the τ_3 to the c_1 which has lower blocking value. At the rest of this section, we explain how to find $EEST_{ij}$ and $ELFT_{ij}$ from the job-level precedence graph.

5.2.2 Finding the Expected Earliest Start Time

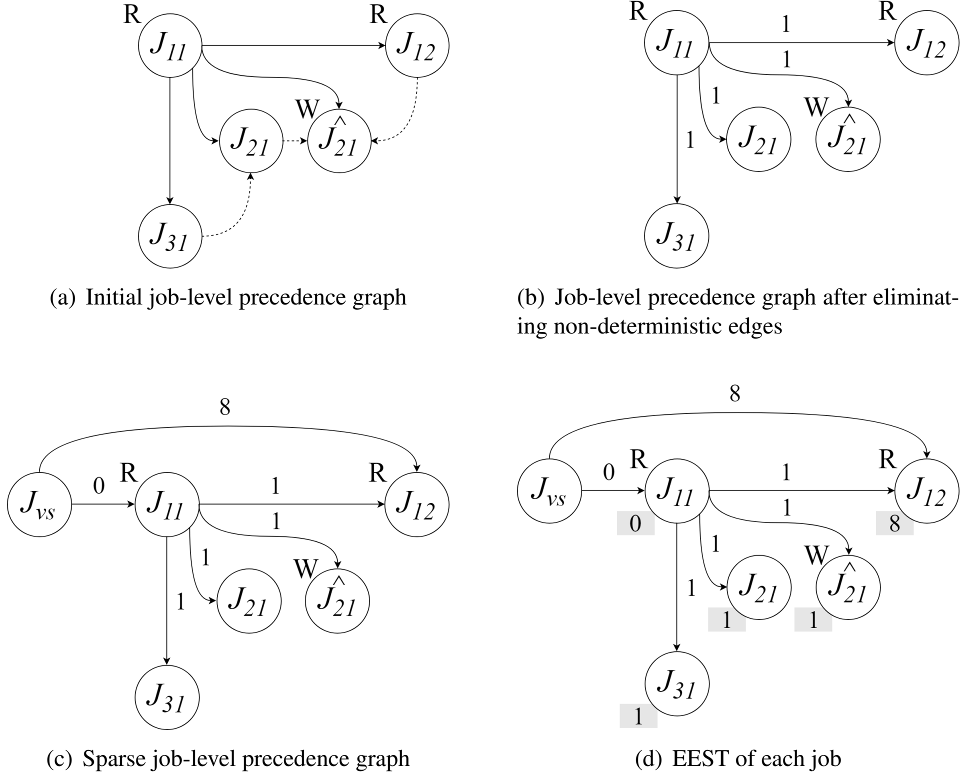
The more precedence relations in the job-level precedence graph make the jobs start more later since they force the successor job to start after all of its predecessor jobs finish. Therefore, we need to use more sparse job-level precedence graph to conservatively expect the $EEST_{ij}$s. To this end, we eliminate the non-deterministic edges, which are not sure to be deterministic or removed during the simulation, from the job-level precedence graph.

After eliminating the non-deterministic edges from the job-level precedence graph, we assign the edge weights to the remaining edges following:

$$w(J_{ij}, J_{kl}) = C_i^{best,sim} \tag{5}$$

where $C_i^{best,sim}$ represents the best case execution time of τ_i on the simulator. Figure 8(a), (b) show this process using the example job-level precedence graph in the Figure 4. Since we assign the edge weights as the best case execution time of the predecessor job, the length of the longest path from the initially scheduled job to the J_{ij} equals to the $EEST_{ij}$ when we assume the infinite number of simulator cores which provides ideally parallelized execution.

The job-level precedence graph after eliminating non-deterministic edges forms a DAG (Directed Acyclic Graph) because it is already proven that the job-level precedence graph with only deterministic edges cannot contain cycle [2]. Since there exists a polynomial time algorithm to find the longest path on the DAG, we can find the $EEST_{ij}$ of each job in polynomial time [16]. However, unlike the normal DAG-formed job-level precedence graph, our job-level precedence graph has a constraint



(a) Initial job-level precedence graph

(b) Job-level precedence graph after eliminating non-deterministic edges

(c) Sparse job-level precedence graph

(d) EEST of each job

Figure 8: Construction of sparse graph and $EEST_{ij}$ of each job for the example job-level precedence graph

about the start time of each job as the Eq. 1 shows. It means that the start time of the J_{ij} who has physical read constraint is affected by not only its predecessor jobs but also its actual start time on the real cyber-side. In other words, although the physical read job does not have any unfinished deterministic predecessor, it cannot be added to the ready queue until its actual start time on the real cyber-side.

To consider such constraint, we add virtual start job J_{vs} to the job-level precedence graph and connect it to the jobs who have physical read constraint. At this step, we additionally connect the J_{vs} to the jobs who do not have any predecessor with zero-weight. These edges allow us to regard the J_{vs} as a single start job of the job-level

precedence graph by collecting all the jobs who might could be initially scheduled. In summary, the weights of newly connected edges are assigned following:

$$w(J_{vs}, J_{ij}) = \begin{cases} min(t_{ij}^{S,real}), & \text{when } J_{ij} \text{ has physical read constraint} \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

As we mentioned at the Section 3, the start time interval $[min(t_{ij}^{S,real}), max(t_{ij}^{S,real})]$ is progressively narrowed during the simulation. i.e., the value of $min(t_{ij}^{S,real})$ keeps increasing. By assigning the least narrowed $min(t_{ij}^{S,real})$ value as the weight of edge from the J_{vs} to the J_{ij} who has physical read constraint, we can force the length of the longest path from the J_{vs} to the J_{ij} to be larger than the earliest start time of J_{ij} on the real cyber-side. Figure 8(b), (c) show this process. After adding the virtual start job and assigning the corresponding edge weights, we find the lengths of the longest path from the J_{vs} to each job as Figure 8(d) shows. The shaded box at the lower-left corner of each job J_{ij} represents the length of the longest path which equals to $EEST_{ij}$.

5.2.3 Finding the Expected Latest Finish Time

Similar with the start time, the more precedence relations in the job-level precedence graph make the jobs finish later. Therefore, we need to use more dense job-level precedence graph to conservatively expect the $ELFT_{ij}$s. To this end, in this time, we regard the non-deterministic edges as the deterministic edges. However, unlike the sparse graph, when we consider both of deterministic and non-deterministic edges, the job-level precedence graph may contain cycle which makes it impossible to define the longest path from the job to the another job. Therefore, we first resolve the cycle by eliminating the one of the non-deterministic edges composing the cycle.

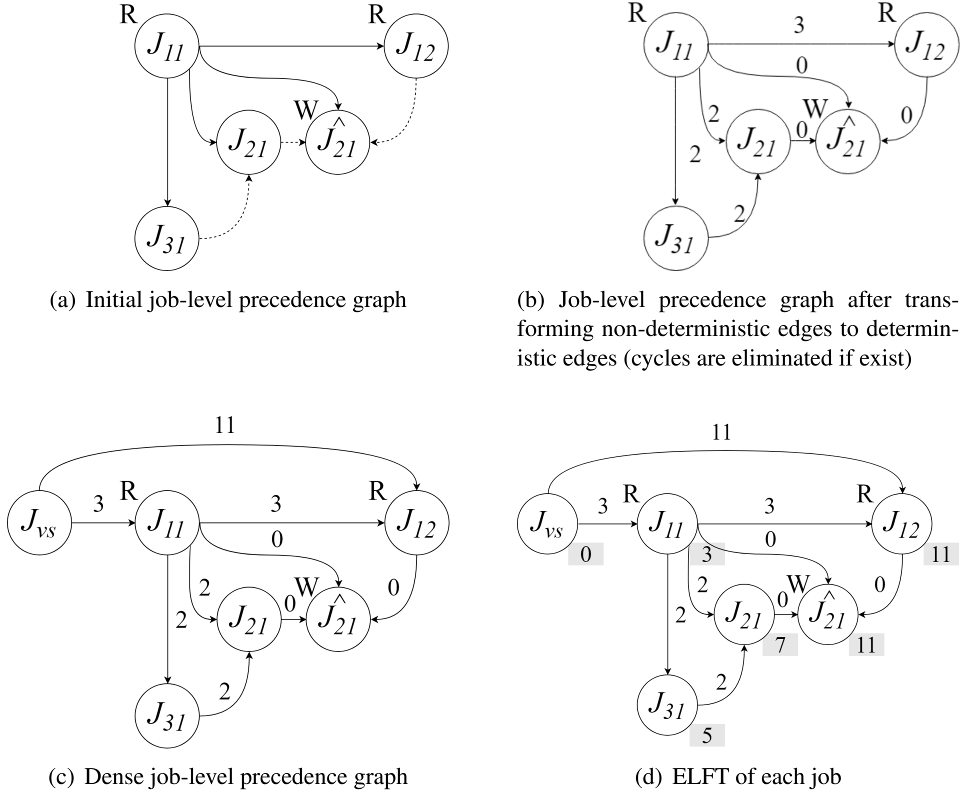


Figure 9: Construction of dense graph and $ELFT_{ij}$ of each job for the example job-level precedence graph

Whenever a job in the job-level precedence graph is finished, the simulator checks below inequality using the narrowed $t_{ij}^{S,real}$, $t_{ij}^{F,real}$ ranges for all of the remaining non-deterministic edges (J_{kl}, J_{ij})s:

$$max(t_{kl}^{S,real}) < min(t_{ij}^{S,real}) \tag{7}$$

If the above inequality holds, the non-deterministic edge (J_{kl}, J_{ij}) becomes deterministic [2]. During the simulation, the value of $max(t_{kl}^{S,real})$ keeps decreasing and $min(t_{ij}^{S,real})$ keeps increasing according to the narrowed $t_{ij}^{S,real}$, $t_{ij}^{F,real}$ ranges. There-

fore, it intuitively implies that the smaller difference between $max(t_{kl}^{S,real})$ and $min(t_{ij}^{S,real})$ makes the non-deterministic edges more likely to be deterministic. From this speculation, we delete the non-deterministic edge who has the largest $max(t_{kl}^{S,real}) - min(t_{ij}^{S,real})$ value among the non-deterministic edges composing the cycle. Since there exist plenty of polynomial time cycle detection algorithms [17], we can resolve the cycles in polynomial time by repeating the deletion of such non-deterministic edge until no more cycle is detected.

After resolving the cycles, we assign the edge weights to the remaining edges following:

$$w(J_{ij}, J_{kl}) = C_k^{worst,sim} \tag{8}$$

where $C_k^{worst,sim}$ represents the worst case execution time of τ_k on the simulator. We also add the virtual start job and its corresponding edges from the J_{vs} to the jobs who have physical read constraint or do not have any predecessor. The weights of edges which are incident with the J_{vs} are assigned following:

$$w(J_{vs}, J_{ij}) = \begin{cases} max(t_{ij}^{S,real}) + C_i^{worst,sim} & \text{when } J_{ij} \text{ has physical read constraint} \\ C_i^{worst,sim}, & \text{otherwise} \end{cases} \tag{9}$$

Unlike the sparse graph, we assigned the edge weights following the worst case execution time of the successor job, and the virtual start job forces the physical read jobs to start and finish as late as possible. i.e., they start at the latest start time on the real cyber-side, $max(t_{ij}^{S,real})$, and are executed for the worst case execution time, $C_i^{worst,sim}$. These assignments allow us to regard the length of the longest path from the J_{vs} to the J_{ij} as the latest finish time of the J_{ij} on the cycle-eliminated job-level precedence

graph when we assume the ideally parallelized execution. Figure 9(a) through (d) show these processes to find $ELFT_{ij}$. $EFLT_{ij}$ for each job is denoted as the shaded box at the lower-right corner of each job. Note that unlike the $EEST_{ij}$, we cannot guarantee the simulated job J_{ij} in the job-level precedence graph finishes before the $ELFT_{ij}$ because we find it after deleting the non-deterministic edge who might could be deterministic during the simulation. In the rest of this paper, $ELFT_{ij}$ means the latest finish time on the cycle-eliminated job-level precedence graph. i.e., the latest finish time on the dense graph. However, we will use the $ELFT_{ij}$ as the approximated finish time of the job J_{ij} in the original job-level precedence graph since we delete the least likely to be deterministic edge when we resolve the cycles.

5.2.4 Weighting the [$EEST_{ij}$, $ELFT_{ij}$] intervals

Our conservative approach to find [$EEST_{ij}$, $ELFT_{ij}$] intervals may leads us to expect too broad intervals which cannot practically predict the task-wise blockings. Since the job J_{ij} can be executed only up to for $C_i^{worst,sim}$ within the interval [$EEST_{ij}$, $ELFT_{ij}$], we weight the each interval as follow:

$$w([EEST_{ij}, ELFT_{ij}]) = \frac{C_i^{worst,sim}}{ELFT_{ij} - EEST_{ij}} \tag{10}$$

Figure 10 shows the resulting weighted intervals and task-wise blocking values for the example job-level precedence graph in the Figure 5. The Figure 10 represents the situation where τ_1 and τ_2 are already mapped to c_1, c_2, respectively, and we are determining which core τ_3 is mapped to. The task-wise blocking between τ_i and τ_j, B_{τ_i,τ_j}, is defined as the sum of their job-wise blocking, $b_{J_{ik},J_{jl}}$:

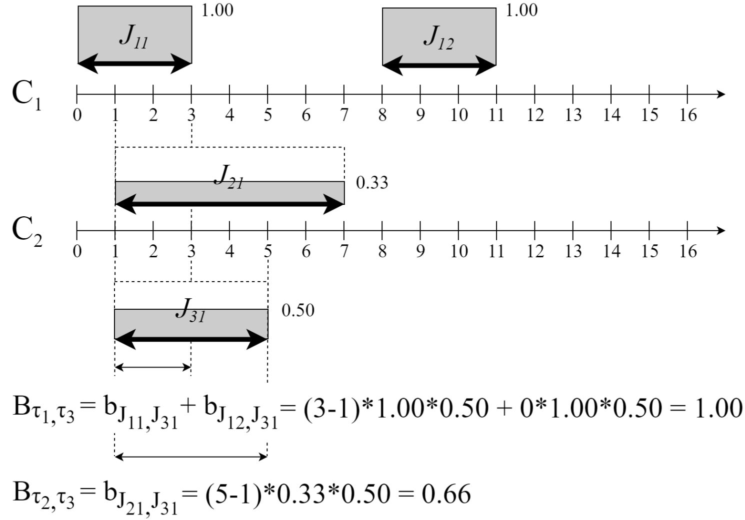


Figure 10: Weighted intervals and task-wise blocking values for the example job-level precedence graph

$$B_{\tau_i,\tau_j} = \sum_{\forall J_{ik} \in \tau_i} \sum_{\forall J_{jl} \in \tau_j} b_{J_{ik},J_{jl}} \tag{11}$$

The job-wise blocking between J_{ik} and J_{jl}, $b_{J_{ik},J_{jl}}$, is defined as the weighted product of interleaved length:

$$b_{J_{ik},J_{jl}} = the\ length\ of\ interleaved\ interval$$
$$* w([EEST_{ik}, ELFT_{ik}]) * w([EEST_{jl}, ELFT_{jl}]) \tag{12}$$

For example, the task-wise blocking between τ_1 and τ_3, B_{τ_1,τ_3}, is defined as the sum of their job-wise blockings $b_{J_{11},J_{31}}$ and $b_{J_{12},J_{31}}$. When we consider $b_{J_{11},J_{31}}$, the interleaved interval is [1, 3] and weights of each interval are 1.00 and 0.50 respectively. Since $[EEST_{12}, ELFT_{12}]$ and $[EEST_{31}, ELFT_{31}]$ are not interleaved each other, the task-wise blocking between the τ_1 and τ_3 equals with $b_{J_{11},J_{31}} = 1.00$.

In summary, our proposed heuristic task partitioning can be represented as Algorithm 1 shows. During the simulation, the simulated job who has no unfinished

Algorithm 1 Proposed task partitioning algorithm

```
1: 𝕋 ← {τ1...τn} // set of tasks
2: ℂ ← {c1...cm} // set of cores
3: 𝕌 ← {U1 = 0...Um = 0} // mem. usage of each core
4: ℙ ← {P1 = ∅...Pm = ∅} // task partition of each core
5: for τi ∈ 𝕋 do
6:     core_min ← −1
7:     block_min ← ∞
8:     for Pj ∈ ℙ do
9:         if Uj + MEM_τi > local cache size of cj then
10:            continue
11:        end if
12:        block ← 0
13:        for τk ∈ Pj do
14:            block = block + B_τi,τk
15:        end for
16:        if block < block_min then
17:            block_min ← block
18:            core_min ← j
19:        end if
20:    end for
21:    if core_min = −1 then
22:        Task partitioning failed!
23:    else
24:        U_core_min = U_core_min + MEM_τi
25:        P_core_min = P_core_min ∪ {τi}
26:    end if
27: end for
```

deterministic predecessor and does not violate the constraint in Eq. 1 is added to the ready queue of its pre-partitioned core. Similar with the previously proposed simulation method at the Section 3, the jobs partitioned into the each ready queue are scheduled following EDF scheduling policy. i.e., partitioned-EDF. As we mentioned before, if all the jobs of partitioned tasks can be scheduled without any deadline miss, we can say that the cyber-side of an automotive system is correctly simulated.

6 Evaluation

To evaluate our proposed approach, we measured the "simulatability" of our simulation method using randomly synthesized cyber-sides of an automotive system. i.e., how many of them are correctly simulated. In the rest of this paper, by "cyber-side", we mean the cyber-side of an automotive system which is similarly given as the Figure 2.

6.1 Simulatability according to the number of cores

At first, we synthesized 9,000 random cyber-sides. Each cyber-side is synthesized as follows. The number of ECUs is determined from $uniform[3,10]$. The number of tasks on each ECU is fixed as 5. Out of all the tasks in each cyber-side, 20% of them read data from the physical-side. Similarly, another 20% of them write data to the physical-side. The data producer/consumer relations among the tasks are randomly configured, but the total number of producer/consumer relations does not exceed the number of tasks in each cyber-side. For each task τ_i, its task parameters are randomly generated as follows. Its task period P_i is randomly selected from $\{10,20,25,50,100\}$ msec while the offset Φ_i is assumed as zero. The worst case execution time $C_i^{worst,real}$ is determined from $uniform(0,10]\%$ of the P_i and the best case execution time $C_i^{best,real}$ is determined from $uniform(0,100]\%$ of the $C_i^{worst,real}$. All the synthesized tasks are assumed to be scheduled following RM scheduling policy. Since the maximum utilization of each ECU cannot exceed L&L(Liu and Layland) utilization bound, 69%[18], we can assume that the set of tasks on each ECU is schedulable at the real cyber-side. Lastly, for all the tasks in each cyber-side, we

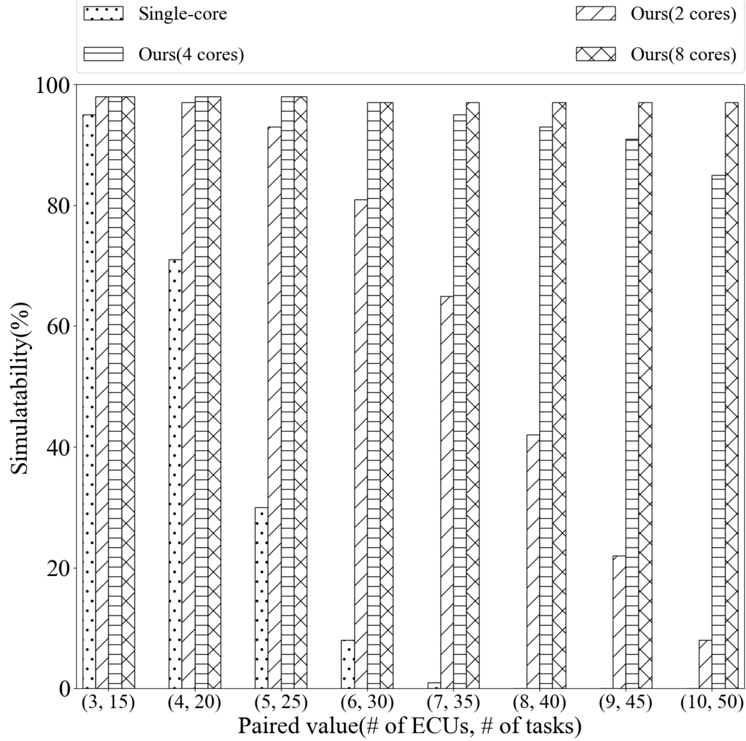


Figure 11: Simulatability according to the number of simulator cores

assume the following simple execution time mapping function:

$$e_{ij}^{sim} = \frac{e_{ij}^{real}}{3} \tag{13}$$

For such synthesized cyber-sides, we measured the simulatability during the ten hyper periods for each cyber-side as changing the number of simulator cores as Figure 11 shows. For fair comparison, we assumed that the memory usage of each task is extremely small, so even all of them can be fit into a local cache of a core. By assuming this, we can focus on only the efficiency of our task partitioning algorithm. i.e., Smallest-blocking-first. We can observe that the simulatability of **Single core**,

the baseline, drops down to 0% when the number of ECUs and tasks pair is (8, 40). On the other hand, our proposed approach using 8 cores, **Ours(8 cores)**, has 97% of simulatability on the same number of ECUs and tasks. Furthermore, by comparing **Ours(2 cores)**, **Ours(4 cores)**, and **Ours(8 cores)**, we can also see that our proposed approach scalably schedules the more ECUs and tasks in line with the increasing number of simulator cores.

6.2 Simulatability according to the partitioning method

Secondly, we synthesized another 9,000 random cyber-sides to compare our proposed approach with other task partitioning heuristics. Each cyber-side is synthesized in the same way with the experiment in the Figure 11 but in this time, the number of ECUs is determined from $uniform[10,17]$. We assumed the small memory usage tasks as in the previous experiment and compared our task partitioning algorithm with following commonly used heuristics:

- **Worst-fit-first:** It considers only memory constraint. It places the new task in a core where it fits loosest. i.e., a core who has the largest remaining local cache size after placing MEM_{τ_i}
- **Smallest-utilization-first:** It considers only utilization of each core. It places the new task in a core which has the smallest utilization where the utilization of a core c_i, $Util_{c_i}$, is defined as below:

$$Util_{c_i} = \sum_{\forall \tau_j \text{ mapped to } c_i} \frac{C_j^{worst,real}}{P_j} \tag{14}$$

Figure 12 shows the result of the experiment. We can see that **Ours** always

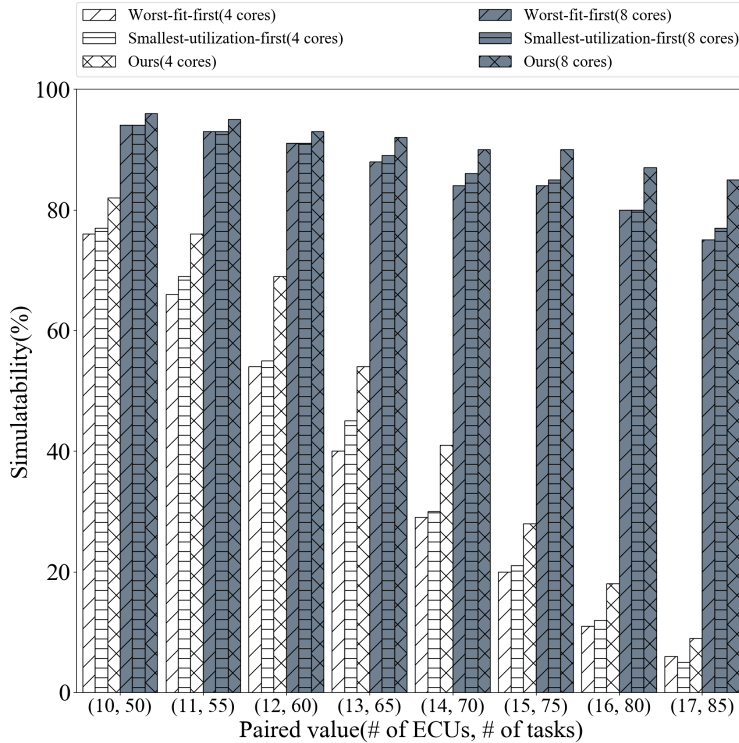
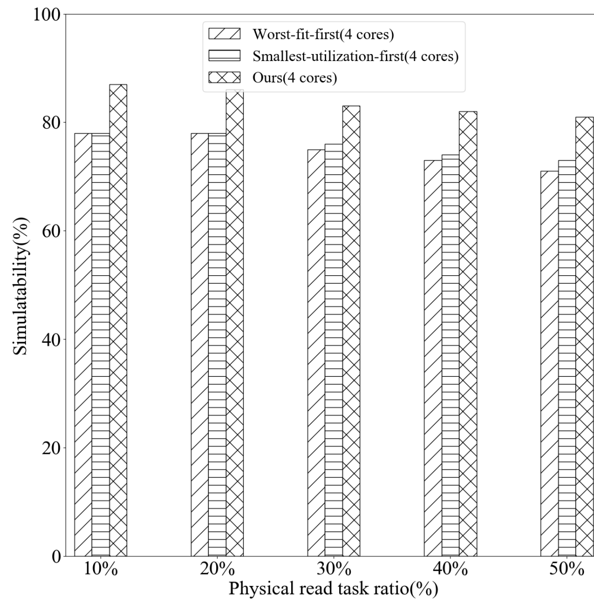


Figure 12: Simulatability compared to the other task partitioning heuristics

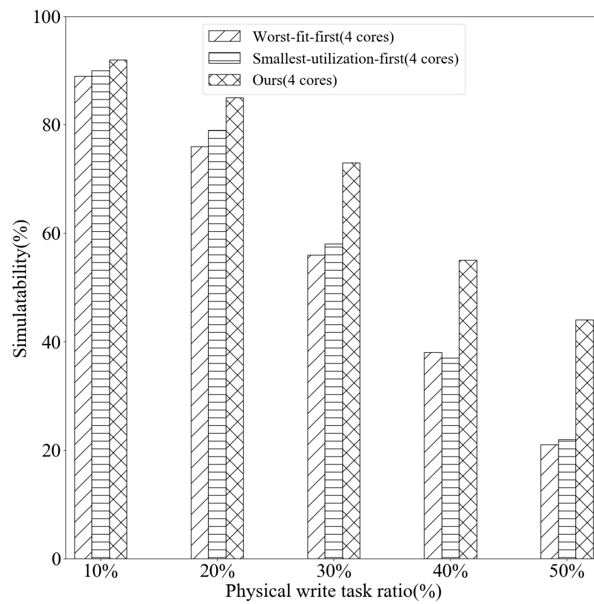
has better simulatability than **Worst-fit-first** and **Smallest-utilization-first** in either **4 cores** and **8 cores**. By comparing **Ours** with **Worst-fit-first** and **Smallest-utilization-first**, we could confirm that Smallest-blocking-first approach has meaningful improvement in comparison with the other heuristics which do not consider the existence of the job-level precedence graph.

6.3 Simulatability according to the physical read/write task ratio

In order to more deeply investigate the simulatability of our approach according to the characteristics of the given cyber-side, we additionally synthesized another 10,000 cyber-sides again. The experiment was conducted in the same conditions with the



(a) Simulatability as changing the physical read task ratio



(b) Simulatability as changing the physical write task ratio

Figure 13: Simulatability according to physical read/write task ratio

previous experiment but in this time, we fixed the number of ECUs as 10 and varied their physical read/wirte task ratios from 10% to 50%.

Figure 13(a) shows the simulatability according to the physical read task ratio. At this experiment, we fixed their physical write task ratio as 20% and varied their physical read task ratio from 10% to 50%. Similar with the before experiments, **Ours** always has better simulatability than others. Since the more physical read tasks in the cyber-side imply the more physical read constraint in Eq. 1, the higher physical read task ratio leads the more jobs in the job-level precedence graph to wait its actual start time on the real cyber-side although they do not have any unfinished deterministic predecessors. We can see such influence through the declines of the simulatabilities according to the physical read task ratio in **Ours**, **Worst-fit-first**, and **Smallest-utilization-first**. However, unlike the other heuristic task partitioning algorithms, Smallest-blocking-first considers the $[EEST_{ij}, ELFT_{ij}]$ intervals which reflect the actual start time on the real cyber-side. The noticeable simulatability gap between **Ours** and others for all physical read task ratios shows such consideration in our approach efficiently handles the physical read tasks.

Similarly, Figure 13(b) shows the simulatability according to the physical write task ratio while the physical read task ratio is fixed as 20%. Since the more physical write tasks in the cyber-side imply the more physical write constraint in Eq. 2, the higher physical write task ratio forces the more jobs in the job-level precedence graph to finish before their actual finish time on the real cyber-side. We can see this tendency through the decreasing simulatabilities of **Ours**, **Worst-fit-first**, and **Smallest-utilization-first** according to the physical write task ratio. We can also validate that our consideration about the $[EEST_{ij}, ELFT_{ij}]$ efficiently handles the physical write

tasks by the increasing simulatability ratio between **Ours** and other heuristic algorithms.

7 Conclusion

This paper proposes the multicore extension of previously proposed functionally and temporally correct single core simulator. The proposed approach consists with two parts: (1) memory constraint for keeping the key ideas of single core simulator and (2) heuristic task partitioning algorithm that aims to minimize the task-wise blocking. By introducing (1), we could guarantee the correct working of the execution mapping functions without any precisely designed isolation technique. We also showed that our memory constraint is not too strict to be satisfied by focusing on the practical usecases of the automotive system tasks. Since our derived memory constraint makes our problem as NP-Complete, we proposed the heuristic algorithm to partition the tasks into the cores. Our heuristic algorithm is empirically validated through the experiments using plenty of synthesized cyber-sides.

The followings are our future works to cover the limitations and extend the proposed approach:

- **Another control knobs to precisely expect the actual execution scenarios of the cyber-side:** In this dissertation, we proposed the conservative method for expecting the start/finish time of each job. In the future, by studying another knobs to precisely expect the real cyber-side behaviors, we expect to improve the simulatability of our proposed approach.

- **Accurate execution time mappings from the simulator environment to ECU environment:** Our concept of correctness is based on the execution time mappings. In the future, by studying more accurate execution time mappings between different machines, we expect to lay the firm foundation of our proposed

approach.

- **Simulation for more complex cyber-physical systems:** In this dissertation, we could simply evade multicore interferences by focusing on the automotive system tasks which have small memory usage. In the future, by studying another method for evading multicore interferences, we expect to extend our coverage of the simulation to more complex cyber-physical systems which normally have immense memory usage and super-large computation amount.

References

[1] Simulink. *version 8.4.0.150421 (R2014b)*. MathWorks Inc., Natick, Massachusetts, 2014.

[2] Kyoung-Soo We, Seunggon Kim, Wonseok Lee, and Chang-Gun Lee. Functionally and temporally correct simulation of cyber-systems for automotive systems. In *Real-Time Systems Symposium (RTSS), 2017 IEEE*, pages 68–79. IEEE, 2017.

[3] Hyejin Joo, Kyoung-Soo We, Seunggon Kim, and Chang-Gun Lee. An end-to-end tool for developing cpss from design to implementation. 2016.

[4] dSPACE. *version 8.4.0.150421 (R2014b)*. dSPACE GmbH., Wixom, Michigan, 2018.

[5] Kyoung-Soo We. *Functionally and Temporally Correct Simulation for Cyber-Physical Systems*. PhD thesis, Seoul National University, 2017.

[6] Intel. Core i7-9700k. https://www.intel.com/content/www/us/en/products/processors/core/i7-processors/i7-9700k.html, 2018. Accessed 1 Nov. 2018.

[7] Infineon. Tricore 27x. https://www.infineon.com/cms/en/product/microcontroller/32-bit-tricore-microcontroller/aurix-safety-joins-performance/aurix-family-tc27xt/, 2018. Accessed 1 Nov. 2018.

[8] Lui Sha, Marco Caccamo, Renato Mancuso, Jung-Eun Kim, Man-Ki Yoon, Rodolfo Pellizzoni, Heechul Yun, Russel Kegley, Dennis Perlman, Greg Arun-

dale, et al. Single core equivalent virtual machines for hard real—time computing on multicore processors. Technical report, 2014.

[9] Abusayeed Saifullah, David Ferry, Chenyang Lu, and Christopher Gill. Real-time scheduling of parallel tasks under a general dag model. 2012.

[10] Heechul Yun, Renato Mancuso, Zheng-Pei Wu, and Rodolfo Pellizzoni. Palloc: Dram bank-aware memory allocator for performance isolation on multicore platforms. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2014 IEEE 20th*, pages 155–166. IEEE, 2014.

[11] Moinuddin K Qureshi and Yale N Patt. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *Microarchitecture, 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium on*, pages 423–432. IEEE, 2006.

[12] Sophie Stern and Cigdem Gencel. Embedded software memory size estimation using cosmic: A case study. In *Int'l Workshop on Software Measurement (IWSM)*, volume 39, 2010.

[13] Intel. Core i7-3610qm. `https://ark.intel.com/products/64899/Intel-Core-i7-3610QM-Processor-6M-Cache-up-to-3-30-GHz-`, 2012. Accessed 8 Nov. 2018.

[14] Michael R Garey and David S Johnson. Computers and intractability: A guide to the theory of npcompleteness (series of books in the mathematical sciences), ed. *Computers and Intractability*, 340, 1979.

[15] Hoon Liong Ong, Michael J Magazine, and TS Wee. Probabilistic analysis of bin packing heuristics. *Operations Research*, 32(5):983–998, 1984.

[16] Ei Ando, Toshio Nakata, and Masafumi Yamashita. Approximating the longest path length of a stochastic dag by a normal distribution in linear time. *Journal of Discrete Algorithms*, 7(4):420–438, 2009.

[17] Gabriel Nivasch. Cycle detection using a stack. *Information Processing Letters*, 90(3):135–140, 2004.

[18] Chung Laung Liu and James W Layland. Scheduling algorithms for multi-programming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

요약(국문초록)

본 논문에서는 멀티 코어 시뮬레이터를 활용하여 자동차 사이버-물리 시스템의 사이버 시스템을 기능적/시간적으로 정확하게 시뮬레이션하기 위한 방법을 제시한다. 앞선 연구에서는 시스템의 기능적 행태뿐만 아니라 태스크의 가변 수행 시간, 자원 선점 등과 같은 시간적 행태 역시 함께 정확히 모사하기 위한 새로운 시뮬레이션 기법들이 제안되었다. 앞선 연구에서 제안된 시뮬레이션 기법들이 싱글코어 시뮬레이터만을 가정하고 있다는 점에 착안하여 본 논문에서는 정확한 시뮬레이션을 보장하기 위해 제안되었던 기존 연구의 주요 아이디어를 모두 유지하면서 싱글코어 시뮬레이터를 멀티코어 시뮬레이터로 확장한다. 제안하는 방법에서는 각 태스크의 메모리 사용량과 근사화 된 태스크 간 블로킹 값을 기반으로 시뮬레이션 대상 태스크에 대한 휴리스틱 태스크 분할 알고리즘을 설계한다. 또한, 임의적으로 생성한 다수의 워크로드를 사용하여 시뮬레이션 성능을 측정하고, 이를 통해 제안하는 방법이 싱글코어 시뮬레이터 및 다른 태스크 분할 알고리즘에 비해 각각 최대 97%p, 15%p의 향상된 시뮬레이션 용량을 갖는 것을 보인다. 결과적으로 제안하는 멀티코어 시뮬레이터는 앞선 연구에서 제안되었던 기능적/시간적 정확성을 동일하게 보장함과 동시에 보다 높은 시뮬레이션 용량을 제공함으로써 전체 자동차 시스템의 시뮬레이션에 효과적으로 활용될 수 있다.

주요어 : 자동차 시스템 시뮬레이션, 실시간 시뮬레이션
학 번 : 2017-21586