Journal of Information Processing Vol.32 1–10 (Jan. 2024)

[DOI: 10.2197/ipsjjip.32.1]

Regular Paper

Communication Overhead Description Schema for Multi Core Processor in Model-based Development

Yue Hou[1] Yutaro Kobayashi[1] Hiroshi Fujimoto[2] Takuya Azumi[1]

Received: December 18, 2023, Accepted: August 1, 2016

Abstract: Autonomous driving systems require a wider range of functionalities than traditional embedded systems, making Model-based Development (MBD) with MATLAB/Simulink essential. This approach allows early simulation and validation, enhancing safety and reducing development time. At the same time, the real-time requirements of multi-tasking for autonomous driving cannot be ignored. Many core processors with computing performance and low power consumption have been developed to satisfy these requirements. However, due to their complexity, developing with many core processors takes time. In MBD, MATLAB/Simulink's automatic code generation is not tailored for many-core systems, leading to the development of a model-based parallelization tool that generates parallelized C code. This tool leverages the software-hardware interface for multi-many-core (SHIM) information—a standard that details the structure of many-core processors and performance metrics. However, SHIM cannot fully demonstrate communication overhead as it describes overhead in instruction units. With this, we propose a new schema to describe communication overhead in API units and an XML split specification to enhance SHIM's reusability. Experimental results show at least a 97.9% improvement in accuracy for estimating communication overhead, over 90% reduction in schema description, and a significant reduction in execution time with our schema.

Keywords: Embedded Systems, Model-based Development, Multi/Many-core

1. Introduction

In recent years, autonomous driving systems [1, 2] have attracted attention for reducing traffic accidents. However, real-time performance is essential for automated driving systems, and high computational power is required. In development of these systems, Model-based Development (MBD) has raised for its functionalities such as early simulation, and validation [3,4]. Furthermore, low power consumption is also required to enable installation in automobiles. Thus, more comprehensive functions are required than those of conventional embedded systems [5]. To meet these requirements, many-core processors are being adopted. Many-core processors offer high computational performance and energy efficiency [6–8]. Efficient utilization of these processors, however, necessitates understanding their complex specifications, which can extend development times.

In MBD, MATLAB/Simulink is noted for its automatic C code generation from models, but it does not support parallelized code generation [9]. Addressing this, the Embedded Multicore Consortium's Model-based Parallelizer (MBP) facilitates the automatic generation of parallelized C code. This integration streamlines simulation and code generation for multi-many-core processors, enhancing development efficiency. Utilizing SHIM (Software-Hardware Interface for Multi-Many-Core) to abstract essential processor information minimizes the reliance on physical hardware [10–13].

The latest version of SHIM [14] currently still lacks an efficient method to describe inter-core communication latency, which poses significant challenges. This deficiency leads to substantial inaccuracies in time predictions within the Model-based Parallelizer (MBP) due to the omission of inter-core communication latency calculations. Furthermore, the lack of a concise communication description necessitates extensive documentation of latency between cores, resulting in increased number of lines for the communication overhead description to more than ten thousand lines.

To address these issues, a revised schema is proposed that measures communication overhead in per-API units for more precise estimates and also simplifies the schema's structure to reduce its volume. This enhancement facilitates ease of use and construction, allowing for a more accurate and comprehensive representation of communication overhead. Additionally, it improves the reliability and efficiency of performance evaluations in multi-core environments.

The main contributions of this paper are as follows:

- This paper proposes a communication overhead schema that is independent of communication libraries and can account for changes in message size.
- This paper shows improvement in results by generating a communication overhead description file with an appropriate message size and using it in existing estimation methods. It

1 Graduate School of Science and Engineering, Saitama University
2 Technology Headquarters, eSOL Co., Ltd

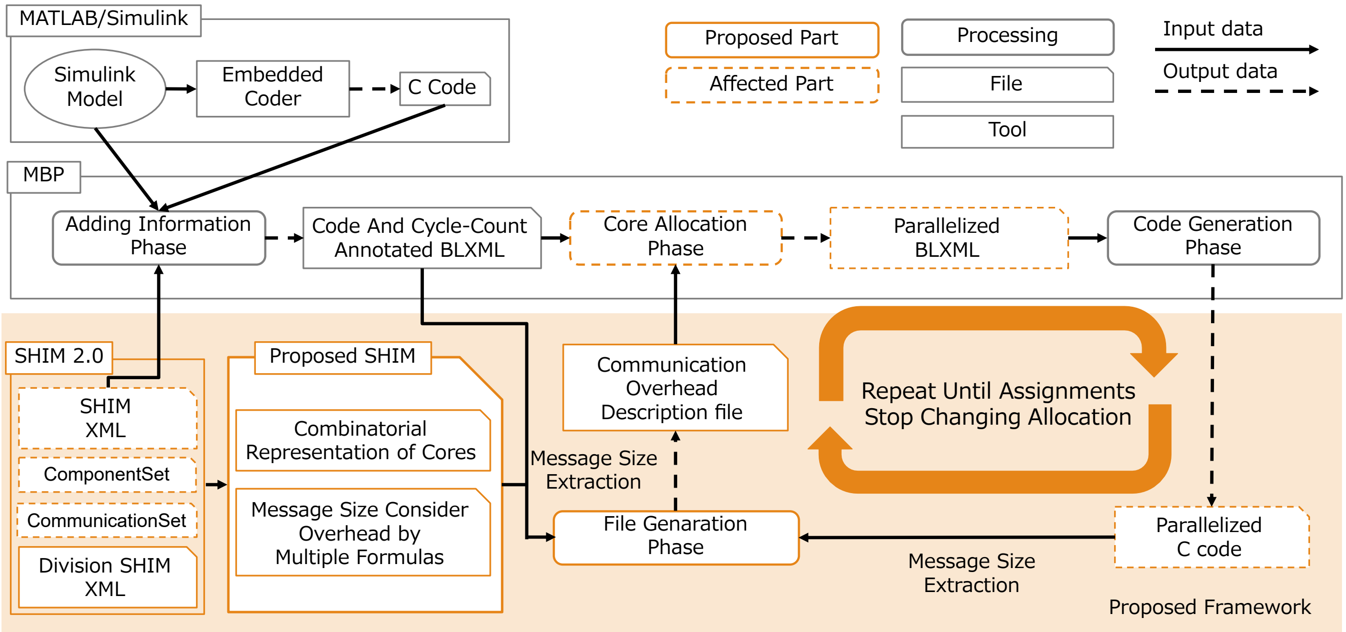

Fig. 1 System model.

demonstrates that the average improvement rate in the error of the estimated communication overhead is at least 97.9%.

- This paper proposes a new XML split specification to reduce the amount of description and the description of cores and combinations of cores by more than 90%, to reduce the burden on the users of SHIM.

The remainder of this paper is organized as follows. Section 2 describes the system model. Next, Section 3 shows the approach. Section 4 discusses the process of applying the proposal schema to MBP. Then, Section 5 shows the experimental results. Section 6 discusses related work. Finally, Section 7 concludes this paper.

2. System Model

This section introduces SHIM, MBP, and the system model shown in **Fig. 1**. The proposed framework is divided into two major parts. The first is a proposal for a schema that describes communication overhead based on per-API measurements. The second is to propose a method to generate a communication overhead description file from the proposed schema to apply the proposed schema to MBP.

2.1 SHIM

SHIM is an IEEE standard interface for describing hardware and software information of multi-many-core processors. SHIM is in XML format and has a tree structure consisting of five upper-level components. Two of the most important are described here. The current latest version is SHIM 2.0. The creation of SHIM XML by hardware providers is essential for software tool usage, yet not all providers supply this XML. This limitation could hinder hardware adoption. To address this, freely available Reference authoring tools, accompanied by specifications, allow users with access to technical manuals and hardware (such as simulators or evaluation boards) to generate SHIM XML for most multi-many-core systems, thus enhancing accessibility and integration.

2.1.1 ComponentSet

ComponentSet is a component representing complex processor core clusters and hardware boards. To represent them, three components are used: *MasterComponent* for processors and accelerators, *SlaveComponent* for the relationship among *MasterComponent*, memory blocks, and memory subsystems. Lastly, *Cache* denotes the cache.

2.1.2 CommunicationSet

This component defines the communication method and overhead between cores by describing the combination of cores and message size. All possible communications should be described in this component. The overhead described is the overhead of the communication only.

2.2 MBP

MBP is a model-based parallelization tool that can be used with MATLAB/Simulink. MBP takes a Simulink model and SHIM as input and can automatically generate parallelized C code and estimate execution time from the model.

2.2.1 Adding Information Phase

MBP extracts the block information from the Simulink model. Next, MBP splits the C code generated by the Embedded Coder into code blocks and generates BLXML with code annotations. Finally, MBP extracts hardware information from the SHIM XML and uses processor latency to estimate the performance of each Simulink block.

2.2.2 Core Allocation Phase

In this phase, MBP generates the mapping information using cycle count annotated BLXML generated in *Adding Information phase*. In addition, incorporating the communication overhead description file enables allocation that considers communication overhead.

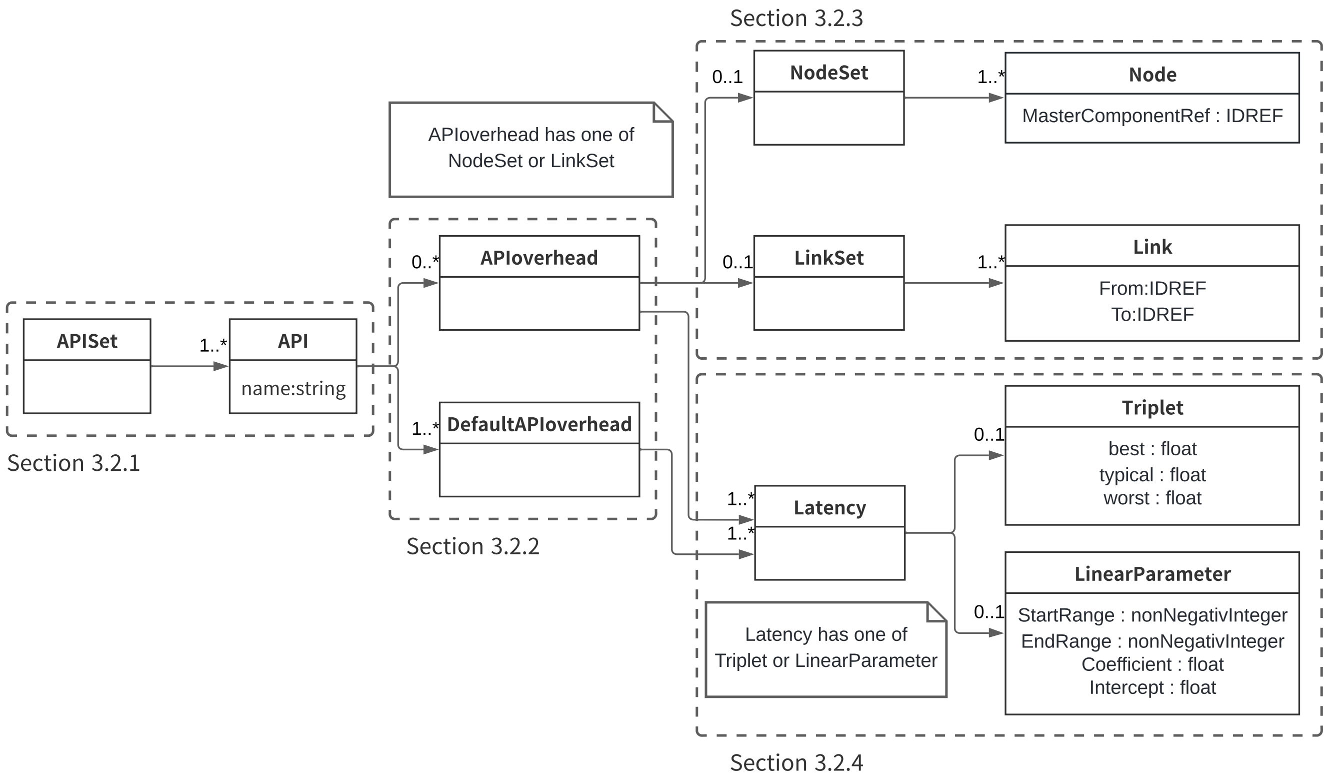

Fig. 2 The proposed schema.

Table 1 OS and APIs investigated

Function	MPI [15]	MCAPI [16]	eMCOS [17]	ROS [18]
Initialize	✓	✓		✓
Finalize	✓	✓		✓
Get node ID	✓	✓	✓	
Get node name			✓	✓
Create node		✓	✓	✓
Delete node	✓		✓	✓
Exit node		✓	✓	
Message send	✓	✓	✓	✓
Message receive	✓	✓	✓	✓

2.2.3 Code Generation Phase

This phase aims to generate the parallelized C code from parallelized BLXML. MBP follows the parallelized BLXML, which includes core assignments and reconstructs the C code for each assigned core to generate parallelized C code.

3. Approach

An implementation of a schema that can describe software overhead will be described. In addition, this section introduces a method to incorporate the proposed schema into SHIM 2.0. Note that this paper does not explain the arguments and use cases that led to the implementation. For more information, please refer to the previous paper [19].

3.1 Design of Proposed Schema

The design of the proposed schema, which can describe software overhead per API, is described. Software overhead can be described more accurately than in SHIM 2.0 because software overhead can be described in units of API. In addition, the proposed schema can be incorporated into SHIM 2.0.

3.1.1 Goals of Proposed Schema

The goal is to create an additional schema for describing software overhead information independent of communication libraries. This schema should be able to describe each of the API overheads involved in communication. The proposed schema is also intended to be incorporated into SHIM 2.0.

3.1.2 Select API Required for Communication

The proposed schema is designed to allow the overhead to be described on a per-API basis to measure the communication's performance accurately. One reason is that the existing SHIM has difficulty expressing the communication overhead of the API. Another reason is that many common APIs are easy to standardize, as shown in **Table 1**.

3.2 API Overhead Representation in Proposed Schema

This subsection describes each element of the proposed schema shown in **Fig. 2**. Note that, as noted above, the discussion that led to the implementation is not presented in this paper.

3.2.1 APISet and API

APISet is the top element of the proposed schema, and the multiplicity of *APISet* to *API* is at least one. *API* is used to distinguish which API is being represented by the name attribute when representing the API overhead.

3.2.2 APIoverhead and Defaultoverhead

APIoverhead and *Defaultoverhead* are elements that summarize API overhead. *Defaultoverhead* defines the basic overhead

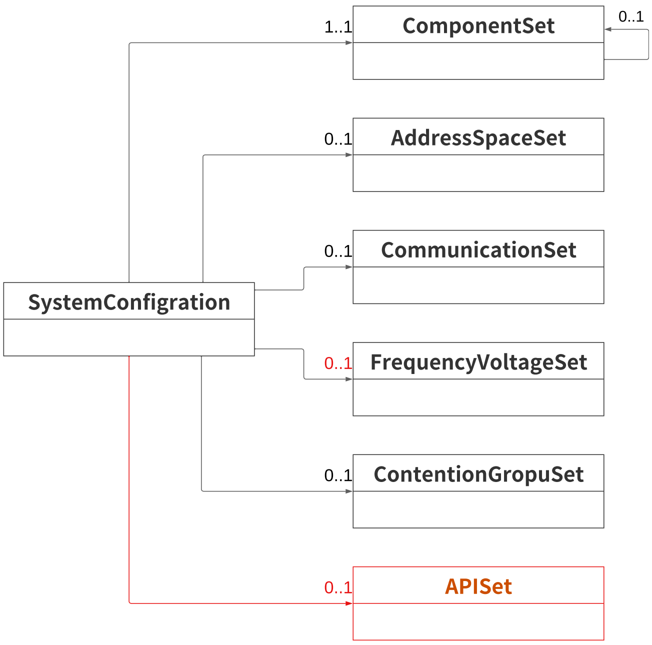

Fig. 3 Overall after incorporating the proposed schema.

of each API on the target processor, and *APIoverhead* defines the API overhead for different performance. In this way, the amount of description can be greatly reduced.

3.2.3 NodeSet, Node, LinkSet, and Link

NodeSet, Node, LinkSet, and Link represent the core or combination of cores on which the API is executed. The proposed schema adopted the idea of grouping *Node* and *Link* by *NodeSet* and *LinkSet* respectively. Each *Node* and *Link* defines either one or a combination of two operating cores. Therefore, combinations belonging to the same *NodeSet* and *LinkSet* have the same API overhead. Note that in [19], Node is Core and Link is Connection, names were changed for consistency with SHIM 2.0.

3.2.4 Latency, Triplet, and LinearParameter

Triplet can define the fixed overhead with three parameters: worst, typical, and best. On the other hand, *LinearParameter* contains the parameters needed to calculate the varying overhead using "linear formulas." Define the coefficient and intercept as necessary parameters for the linear equation. In this paper, the least-squares method is used to compute these two parameters. Furthermore, the proposed schema divides linear expressions by range, allowing for the use of appropriate parameters when calculating overhead. *Latency* is the element provided to select these two exclusively. The mechanism for exclusive selection is similar to the way *APIoverhead* selects *NodeSet* and *LinkSet*.

3.3 Proposed Schema is incorporated into SHIM 2.0

This section describes how the proposed schema is incorporated into SHIM 2.0. In addition, the new specification of the partitioning method is described.

3.3.1 Incorporation of proposed schema into SHIM 2.0

Since the proposed schema is designed with SHIM 2.0 in mind, the proposed schema can be incorporated without modification. Next, the modifications to SHIM 2.0 that are necessary to incorporate the proposed schema are described. The entire modified schema is shown in **Fig. 3**. The multiplicity of *FrequencyVoltageSet* is set to zero or higher because the XML split specification, which is explained in Section 3.3.2, is involved. In addition, although the addition of *APISet* eliminates the role of *CommunicationSet*, *CommunicationSet* is not deleted and is deprecated.

The elements of SHIM 2.0 that are changed by the addition of *APISet* are described. The element to be changed is *Performance*. Previously, the performance had a *Pitch* element and a *Latency* element. These two elements inherited *AbstractPerformance*. After the change, *Pitch* and *Latency* have been modified to select two elements, *Triplet* and *LinearParameter*. For more information, please refer to the previous paper [19].

3.3.2 XML Split Specification to SHIM 2.0

Here, a new XML split specification for SHIM 2.0 is described. The proposed XML split specification aims to improve the reusability of the SHIM schema. The proposed XML split specification utilizes *ComponentSet*, which represents the core of SHIM, as the division unit because *ComponentSet* is the main element of the SHIM schema and is reused frequently. In addition, set the multiplicity of *FrequencyVoltageSet* to "0" or greater for the XML partitioning specification. In this way, XML with only *ComponentSet* can be created. The XML schema uses the ID/IDREF attribute for the reference relation; the ID/IDREF attribute indicates the reference relation between two elements and must be unique in the same XML. Here, an element with the ID attribute can be referenced by another element with the IDREF attribute; if the ComponentSet is split, this ID/IDREF attribute reference relationship is broken.

If the reference relationship is broken by splitting, not only the broken reference relationship but also other reference relationships will not function properly. Therefore, the reference relations in the undivided part were to be maintained by deleting the broken reference relations.

On the other hand, when reusing split XML, a problem arises when composing the main XML (XML containing essential information such as memory access information) and split XML (split *ComponentSet*), which results in duplicate IDs. This problem was solved by assigning a rule to the *ID* attribute to avoid duplication as much as possible. The rules are as follows: "short name of element_file name_random number." This approach eliminates the possibility of duplicate *ID* to the greatest extent possible. The reason for not eliminating duplicates completely is that centralized management of *ID* would be too costly.

ComponentSet can be split using these two ideas. However, repairing referential relations must be considered when reusing the split *ComponentSet* because the referential relations have been deleted to solve the first problem. The repair process is divided into *Cache* and *FrequencyVoltageSet*, which ComponentSet references. As for the *Cache* reference, only one level outward *Cache* can be referenced by SHIM definition and can be repaired automatically. *FrequencyVoltageSet*, on the other hand, is not constrained by references and must be repaired manually.

Algorithm 2 Get Message Size for Overhead Description File

1: **Input:** Simulink model, BLXML
2: **Output:** Message size MS
3: $MS_L \leftarrow$ List of message sizes for signal lines from BLXML
4: **if** MS_L are all equal **then**
5: $\quad MS \leftarrow$ average of MS_L
6: **else**
7: $\quad MS_T \leftarrow$ average of MS_L
8: $\quad MS_P \leftarrow \{\}$
9: $\quad$ **for** $i = 1$ to $2\times$ length of MS_L **do**
10: $\qquad$ Perform core allocation using MS_T
11: $\qquad MS_E \leftarrow$ Extract average of message size from parallelized C code
12: $\qquad$ **if** MS_E equals MS_T **then**
13: $\qquad\quad MS \leftarrow MS_T$
14: $\qquad\quad$ **return** MS
15: $\qquad$ **else if** MS_T is in MS_P **then**
16: $\qquad\quad MS \leftarrow$ average of MS_P
17: $\qquad\quad$ **return** MS
18: $\qquad$ **else**
19: $\qquad\quad$ Append MS_E to MS_P
20: $\qquad\quad MS_T \leftarrow$ average of MS_E
21: $\qquad$ **end if**
22: $\quad$ **end for**
23: $\quad MS \leftarrow$ average of MS_P
24: **end if**

Algorithm 1 Overhead Description File Generation

1: **Input:** Number of cores N, Message size MS
2: **Output:** Overhead Description File
3: $A \leftarrow$ Matrix of size $N \times N$
4: **for** $i = 1$ to N **do**
5: $\quad$ **for** $j = 1$ to N **do**
6: $\qquad$ **if** $i = j$ **then**
7: $\qquad\quad a_{i,j} \leftarrow 0$
8: $\qquad$ **else if** (i, j) belongs to a Link **then**
9: $\qquad\quad$ Reference Link at APIoverhead for $a_{i,j}$
10: $\qquad\quad$ Get LinearParameter for $StartRange \leq MS < EndRange$
11: $\qquad\quad$ Get coefficient as $Coef_{i,j}$ and intercept as $Inte_{i,j}$
12: $\qquad$ **else**
13: $\qquad\quad$ Reference Defaultoverhead.
14: $\qquad\quad$ Get LinearParameter for $StartRange \leq MS < EndRange$
15: $\qquad\quad$ Get coefficient as $Coef_{i,j}$ and intercept as $Inte_{i,j}$
16: $\qquad$ **end if**
17: $\qquad a_{i,j} \leftarrow Coef_{i,j} \times MS + Inte_{i,j}$
18: $\quad$ **end for**
19: **end for**
20: Generate Overhead Description File from A

4. Using the Proposal Schema in MBP

This section describes how the proposed schema is used in MBP. First, the generation method of the communication overhead description schema used in MBP is explained. Next, the method for obtaining the message size, a necessary parameter in the generation method, is introduced.

4.1 Generating Communication Overhead Description Files from Proposed Schema

In order for MBP to use SHIM communication overhead information, a communication overhead description file must be created. This is because MBP reads the overhead description file in the core allocation phase and performs core allocation considering the communication overhead. The algorithm shown in **Algorithm 1** is described in the following.

First, the elements of Input are explained. The number of cores is obtained from SHIM. The message size is obtained in the method outlined in Section 4.1.

In Line 3, initialize a $N \times N$-sized matrix A. Each entry $A_{i,j}$ in this matrix represents the overhead of communication from core i to core j. This matrix format allows for a structured representation of communication costs across all core pairs, facilitating efficient calculation and data management. In Line 8, there is no communication between the same cores ($i = j$). Therefore, the overhead is calculated as zero. This assumption simplifies the model by eliminating unnecessary calculations for non-existent self-communications.

Lines 9-16 obtain the parameters needed to calculate the communication overhead between the different cores ($i \neq j$); from the elements corresponding to Link and MS, the coefficients and intercepts are obtained. The coefficient in this context quantifies the incremental increase in communication overhead per unit increase in message size or other relevant metric. The intercept represents the baseline overhead associated with establishing a communication link, irrespective of the data quantity being transmitted.

The coefficients $Coef_{i,j}$ represent the incremental delay or overhead added per unit of message size or other scaling factor, while the intercept $Inte_{i,j}$ represents the baseline overhead that exists even when no data is being transmitted. Calculate the communication overhead using the coefficients and intercept obtained in Line 17. It is particularly useful for quickly estimating communication delays in complex multi-core systems where direct measurement for every possible communication scenario would be impractical.

This procedure generates a communication overhead description file that can be used for allocation and estimation in MBP. The advantage of the proposed method is that the time required to generate the communication overhead description file is short.

4.2 Methods to Determine Message Size for Calculating Communication Overhead

Since the average communication message size in the Simulink model may deviate from the actual situation, the simulation was repeated to determine the communication message size. The message size must be determined to create a communication overhead description file, as discussed in Section 4.1. The algorithm shown in **Algorithm 2** is described in the following.

In Line 3, get the list of signal line message sizes MS_L from

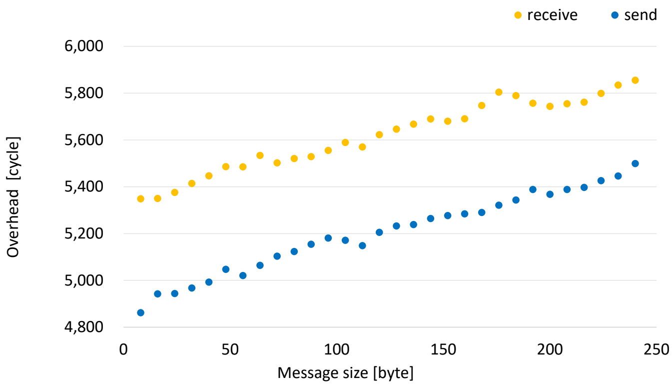

Fig. 4 Overhead of inter-core communication.

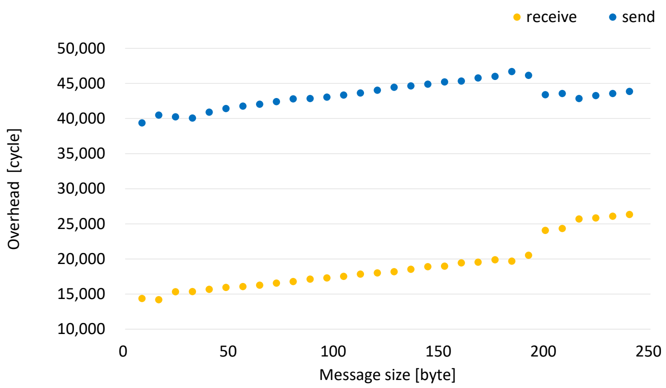

Fig. 5 Overhead of inter-cluster communication.

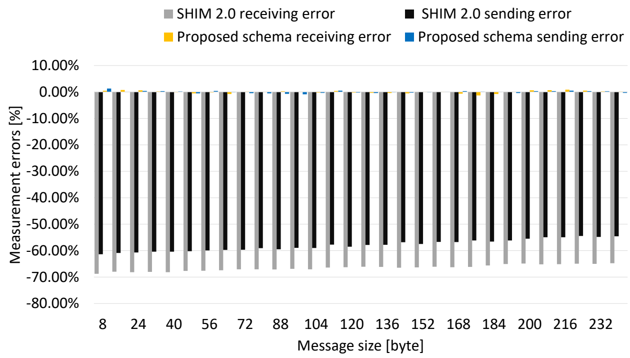

Fig. 6 Comparison of communication overheads (inter-Core).

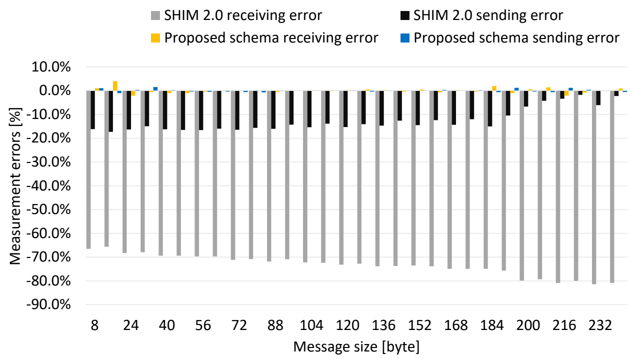

Fig. 7 Comparison of communication overheads (inter-Cluster).

BLXML. In Line 4 - 5, if all elements of MS_L are equal, then the average value of MS_L is directly MS. However, if different message sizes exist, the following procedure is used to determine MS. Initializes the average value of MS_L as MS_T in Line 7. Use MS_T to allocate cores and extract the average message size MS_E from the parallelized C code in lines 10 - 11. If MS_E is equal to MS_T, adopt MS_T as MS and terminate the algorithm in lines 12 - 14. If MS_T exists in the historical list MS_P, adopt the average value of MS_P as MS and terminate the algorithm (Lines 15 - 17). If neither of the above conditions is met, add MS_E to MS_P and compute the new MS_T as the mean value of MS_E. (Lines 18 - 21). If the message size does not remain constant even after looping twice the number of communications, the average of MS_P is used (Line 23). However, since the number of communications is finite, Line 23 is never reached.

5. Evaluation

Then, the proposed schema for the MPPA3-80 Coolidge processor [20] is created, and communication overhead calculated from the schema and the number of descriptive parameters used to calculate communication overhead are compared to those of SHIM 2.0. The difference in communication overhead calculation parameters depending on the number of linear formulas will also be confirmed. Furthermore, the usefulness of using multiple linear formulas is demonstrated by calculating the communication overhead and comparing the results. Finally, the communication overhead description file generated by the proposed method uses MBP to parallelize, and the execution times are compared to show the usefulness of the proposed schema. Additionally, it is important to note that the proposed method is not dependent on the Kalray MPPA3-80 and can be applied to other devices as well, including RISC-V multi-core processors such as the P870 and the P870-A provided by SiFive [21, 22].

5.1 Evaluation Environment

This paper uses the Kalray MPPA3-80 Coolidge processor, a homogeneous processor. MPPA3-80 Coolidge is a clustered many-core processor comprising five clusters, each with 16 cores for a total of 80 cores. The communication overhead of the MPPA3-80 Coolidge processor was measured on an actual device, and the results are shown in **Figs. 4** and **5**. Each measurement was repeated 1,000 times, and the average value of the center 80% of the acquired values was used. The target real-time operating system was eMCOS, and eMCOS functions were utilized to measure the execution time.

5.2 Evaluation of Proposed Schema

Three evaluations were conducted to confirm the effectiveness of the proposed, i.e., evaluations of communication overhead error, error reduction through multiple linear formulas, and description reduction.

5.2.1 Communication Overhead Evaluation

The errors between the communication overhead calculated by the proposed method and SHIM 2.0 and the actual measurement are shown in **Figs. 6** and **7**. In fact, only the communication part is measured because SHIM 2.0 was not able to perform the calculation. This is because the target API was too complex to parse and SHIM 2.0 could not estimate on an instruction-by-instruction basis. On the other hand, the proposed schema calculates com-

Table 2 Parameters used to calculate communication overhead

	Coefficient	Intercept	Link	MessageSize
Send	2.1	5,357.2	Inter-core	All
	31.6	14,266.2	Inter-cluster	Less than 200 bytes
	37.4	12,330.1	Inter-cluster	Greater than 200 bytes
Receive	2.4	4,906.1	Inter-core	All
	59.4	39,495.1	Inter-cluster	Less than 200 bytes
	10.1	41,191.6	Inter-cluster	Greater than 200 bytes

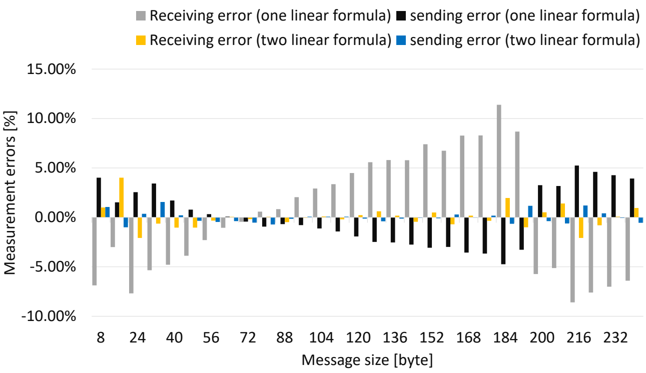

Fig. 8 Comparison of communication overhead (difference in the number of formulas).

munication overhead using the formula $Coef_{i,j} \times MS + Inte_{i,j}$. The parameters used in the communication overhead formula of the proposed schema are shown in **Table 2**. As discussed in Section 3.2.4, this parameter was calculated from the measured values using the least squares method.

A comparison of the results shows that SHIM 2.0 underestimates the communication overhead. This occurred because SHIM 2.0 measures only the communication part and does not consider other related processes, such as memory allocation required for communication. In contrast, the proposed schema measures overhead of other processes per API; thus, the result shows an average improvement rate for the *send* API about 97.9%, and an average improvement rate for the *receive* API about 98.7%. This value is within the SHIM target of 20%, which is sufficient for practical application.

5.2.2 Error Reduction by Multiple Expressions

In the given scenario, two separate linear equations are utilized to approximate the communication overhead for sending and receiving operations, respectively. This differentiation is crucial because the processes involved in sending and receiving data often have distinct characteristics and resource requirements, which can significantly impact communication overhead.

The first linear equation, with a $Coefficient$ of 48.5 and an *Intercept* of 12,999, is used to estimate the overhead for receiving operations. The coefficient here likely represents the incremental overhead per unit of data received, while the intercept might account for fixed costs associated with initiating or terminating a receive operation. The second linear equation, with a $Coefficient$ of 20.0 and an *Intercept* of 40,800, is designed for sending operations. In this case, the lower coefficient suggests that the variable overhead per unit of data sent is less than that of receiving. However, the higher intercept indicates a greater fixed cost, possibly due to the complexities and resources involved in preparing or dispatching data.

By using two different formulas, the model can more accurately reflect the differing nature and costs of sending and receiving data, leading to a more precise and realistic estimation of communication overhead in systems where these activities may vary in frequency, size, and complexity.

The error between the results calculated by the formula $Coef_{i,j} \times MS + Inte_{i,j}$ using the parameters and the actual measured values is shown in **Fig. 8**. The results demonstrate that the error was smaller when multiple linear formulas were used than when only a single linear formula was used. This is because the increase in communication overhead does not fully scale with the message size. If the error is to be further reduced, this can be achieved by setting the range to three and using three linear expressions. Thus, using multiple linear formulas improves the flexibility of the schema.

Table 3 Comparison of the amount of communication overhead

	Existing schema (SHIM 2.0)	Proposed schema
Communication combinations (α)	6,320 combinations	20 combinations
Message size combinations	30 combinations	–
Number of linear formulas (β)	–	two combinations
Number of lines for default overhead description	–	eight lines
Number of lines for individual overhead description	six lines	(β) × six lines + (α) × three lines
Number of lines for the communication overhead description	1,137,600 lines	80 lines

5.2.3 Description Reduction

The results of comparing the amount of description of the SHIM 2.0 and the proposed schema are shown in **Table 3** compares the amount of description. Here, the evaluation target was a Coolidge processor comprising five clusters of 80 cores. The message size ranges from 8-240 bytes (240 bytes is the upper limit of Coolidge processor) in increments of eight bytes for a total of 30 message sizes.

First, SHIM 2.0 requires all communication combinations to be described. Therefore, the number of *Communication combination* was $80 \times 79 = 6,320$ On the other hand, the proposed schema describes inter-core communication as *Defaultoverhead*; thus, only the inter-cluster communication needs to be considered, which is $5 \times 4 = 20$. Next, in SHIM 2.0, the *Message size combination* is 30. On the other hand, in the proposed schema, *The number of linear formulas* is two. In addition, since all inter-core communication is represented by default overhead, there are six lines.

Finally, *The number of lines for individual overhead description* is the calculation that yields SHIM 2.0 to be $6,320{\times}30{\times}6$ for 1,137,600 lines. In contrast, the proposed schema can be represented by $8+2*6+20\times3$ for 80 lines; thus, the proposed schema can reduce the amount of description significantly compared to SHIM 2.0, which can lessen the user's burden when creating or using the SHIM.

Table 4 Comparison of the proposed method and other methods

	MATLAB Simulink	Consider Communication	Parallel Code Generation	Estimation API Overhead	Use SHIM	Improvement SHIM
MBP for CPUs and GPUs [23]	✓	✓	✓			
HS-MBP [24]	✓	✓	✓			
MBP [10]	✓	✓	✓		✓	
RBM & DEOM [25]	✓	✓	✓		✓	
Honda et al. [13]	✓	✓		✓	✓	
Mikami et al. [26]	✓			✓	✓	✓
Kobayashi et al. [19]	✓	✓		✓	✓	✓
This paper	✓	✓	✓	✓	✓	✓

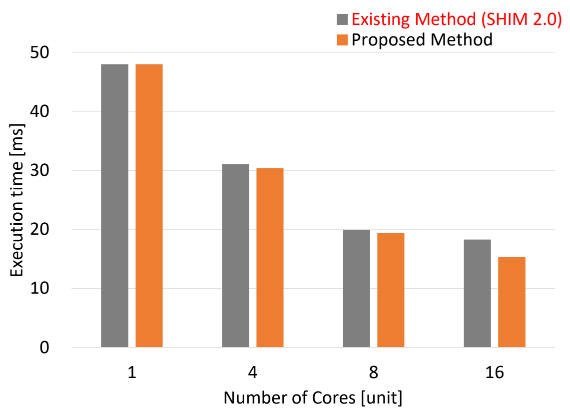

Fig. 9 Comparison of execution times for random models (constant message size).

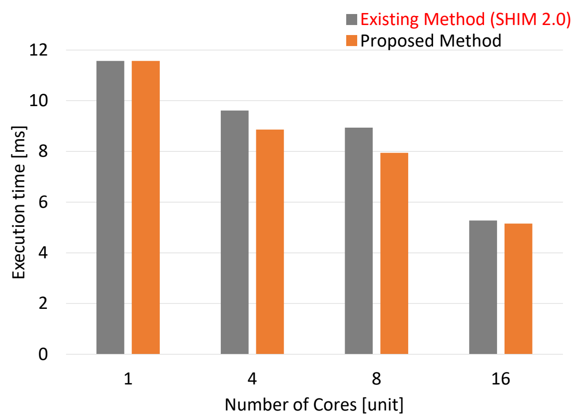

Fig. 10 Comparison of execution times for *map_callback* models.

5.3 Execution Time Evaluation in MBP with the Proposed Schema

To evaluate the usefulness of the proposed schema, a communication overhead description file is generated using the method described in Section 4, and the results of using the file in MBP are presented here. In this evaluation, a random model was generated using the functionality of SLForge [27], and a model representing a part of the actual automatic operation function was used.

First, the parallelization results when using the random model are reviewed. As shown in **Fig. 9**, the results of the execution time for core allocation use both the MBP and proposed approaches. The findings indicate that the execution time decreased when more cores were used, and the execution time was further reduced when implementing the proposed method. Specifically, with 16 cores, a speedup of 1.19 times was achieved, indicating a noteworthy improvement.

Next, the parallelization results when using the models representing a portion of the actual autonomous driving functionality are reviewed. Here, the compared model represented the *map_callback* function in the standard distribution transform [28] of Autoware.Auto. In addition, the models were data parallelized to allow parallelization, and a model of 55,000 treatments was divided into 32 parts. The execution time results for the core allocation using the conventional and proposed methods are shown in **Fig. 10**.

As can be seen, the same as the random model, the execution time is reduced as the number of cores increases, and the execution time is further reduced when the proposed method is applied. The execution time reduction for the eight-core case was less than that for the other cases; however, this was because the increase in overhead time due to the increase in the number of communications was more significant than the speedup due to parallelization. However, since the purpose of this paper is only to improve SHIM, MBP, an existing method, will not be discussed.

In this way, the proposed schema improved the results of the existing method MBP by generating a communication overhead description file. This indicates that the proposed schema is more valuable than SHIM 2.0.

6. Related Work

This section introduces the exsiting studies and compares them with this study in **Table 4**. MBP [23], linear programming (ILP)-based code parallelization for execution platforms with the same number of CPUs and GPUs, was proposed by Zhong et al. This method converts a Simulink model into a DAG and considers task allocation and communication costs to achieve speedup.

The other is HS-MBP [24], an integrated development environment for generating parallelization code and executables for FPGA-based MPSoCs (FP-HSoCs), proposed by Ryota et al. The parallelized C code is automatically generated from Simulink models and converted to a form corresponding to each PE (Processing Element) as an extension of the existing method, MBP.

MBP method for parallelizing Simulink models on heterogeneous multi-core processors was proposed by Zhong et al. [10]. First, groups the blocks hierarchically to form dozens of clusters.

In these clusters, the core allocation is performed in a mixed integer linear programming (MILP) formulation that considers load balancing. Next, block dependencies extend the Model-based parallelization method to the block level.

Two methods to minimize the makespan were proposed by Kojima et al. [25]. The Remapping Blocks Method (RBM) uses the MBP results to remap the blocks to the core. As a result, the critical path can eliminate inter-core communication while maintaining load balancing. Deciding Execution Order Method (DEOM) determines the execution order in which the entire process is completed and is faster. These methods increase the parallelism of blocks compared to conventional methods and speed up processing while distributing the load.

These studies mainly generate parallelized codes and are similar to this study. However, this study differs from existing studies in that it does not propose a parallelization code generation method and improves the input for the generation method. In addition, this study applies to existing studies using SHIM.

The method for estimating the total execution time on a many-core processor of an embedded system developed with the MATLAB/Simulink model in model-based development is proposed by Honda et al. [13]. This study measured various performance information of MPPA2-256 Bostan.

A regression analysis method to estimate the execution cycle of each instruction in LLVM-IR was proposed by Mikami et al. [26]. Two methods are used in this study. A software performance estimation method using SHIM was proposed, including an estimation formula considering finite registers. The second is to estimate the execution cycle for each LLVM-IR instruction stored in SHIM by regression analysis and improve the measurement of LLVM-IR instructions.

These studies mainly estimate the overall execution time. In this respect, this study is similar in that it computationally estimates the communication overhead.

In the previous work [19], we proposed a new schema for describing communication overhead per-API without relying on communication libraries. This approach was assessed through a detailed requirements analysis, use case studies, and instance diagram example evaluation. On the other hand, this paper shows a method to create an actual hardware SHIM based on our proposed approach, along with a method to generate a communication overhead file from this SHIM. Furthermore, the practicality of this method was confirmed by integrating the generated communication overhead file into a parallelization technique. In addition, a new partitioning method was proposed to improve the reusability of the schema further.

7. Conclusion

In this paper, a schema that can be expressed per API was proposed to solve the problem in the existing SHIM schema in terms of expressing communication overhead. The proposed schema improves the error by up to 60% compared to the existing SHIM schema by calculating the communication overhead using multiple linear expressions. In addition, the proposed schema reduces the description amount by greater than 90% compared to the existing SHIM schema by collectively describing the cores and combinations of cores on which the API operates. Moreover, the proposed schema can be incorporated into the existing SHIM schema; thus, a communication overhead description file was generated from the proposed schema and applied to the current tool. The proposed schema was applied to MBP, and the allocation algorithm for clusters achieved a speedup of up to 14%. Furthermore, an XML partitioning specification was proposed to improve the reusability of the existing SHIM schema.

In the future, based on the proposed communication description scheme, more accurate estimations are expected to be realized, which could be utilized in parallelization and contribute to the safety and real-time performance of autonomous driving.

Acknowledgment

This work was supported by JST PRESTO Grant Number JPMJPR21P1.

References

[1] Shinpei Kato, Shota Tokunaga, Yuya Maruyama, Seiya Maeda, Manato Hirabayashi, Yuki Kitsukawa, Abraham Monrroy, Tomohito Ando, Yusuke Fujii, and Takuya Azumi. Autoware on Board: Enabling Autonomous Vehicles with Embedded Systems. In *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, pages 287–296, 2018.

[2] Keita Miura, Shota Tokunaga, Noriyuki Ota, Yoshiharu Tange, and Takuya Azumi. Autoware Toolbox: MATLAB/Simulink Benchmark Suite for ROS-Based Self-Driving Software Platform. In *Proceedings of the 30th International Workshop on Rapid System Prototyping (RSP)*, pages 8–14, 2019.

[3] MathWorks. Automated Driving Toolbox. `https://jp.mathworks.com/products/automated-driving.html`.

[4] Keita Miura, Shota Tokunaga, Noriyuki Ota, Yoshiharu Tange, and Takuya Azumi. Autoware Toolbox: MATLAB/Simulink Benchmark Suite for ROS-based Self-driving Software Platform,. In *Proceedings of the 30th International Workshop on Rapid System Prototyping (RSP)*, pages 8–14, 2019.

[5] Takuya Azumi, Yuya Maruyama, and Shinpei Kato. ROS-lite: ROS Framework for NoC-Based Embedded Many-Core Platform. In *Proceedings of the 33th IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4375–4382, 2020.

[6] Benoît Dupont de Dinechin. INVITED Consolidating High-Integrity, High-Performance, and Cyber-Security Functions on a Manycore. In *Proceedings of the 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–4, 2019.

[7] Matthias Becker, Dakshina Dasari, Borislav Nicolic, Benny Akesson, Vincent Nélis, and Thomas Nolte. Contention-Free Execution of Automotive Applications on a Clustered Many-Core Platform. In *Proceedings of the 28th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 14–24, 2016.

[8] Quentin Perret, Pascal Maurère, Éric Noulard, Claire Pagetti, Pascal Sainrat, and Benoît Triquet. Mapping Hard Real-Time Applications on Many-Core Processors. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems (RTNS)*, pages 235–244, 2016.

[9] MathWorks. MATLAB/Simulink. `http://www.mathworks.com/products/simulink/`.

[10] Zhaoqian Zhong and Masato Edahiro. Model-Based Parallelizer for Embedded Control Systems on Single-ISA Heterogeneous Multicore Processors. In *Proceedings of the 15th International SoC Design Conference (ISOCC)*, pages 117–118, 2018.

[11] IEEE-Standard-Association. SHIM. `https://standards.ieee.org/standard/2804-2019.html`.

[12] Masaki Gondo, Fumio Arakawa, and Masato Edahiro. Establishing a Standard Interface between Multi-Manycore and Software Tools-SHIM. In *Proceedings of the 17th IEEE Symposium on Low-Power and High-Speed Chips and Systems (COOL Chips)*, pages 1–3, 2014.

[13] Kentaro Honda and Takuya Azumi. Performance Estimation for Many-core Processor in Model-Based Development. In *Proceedings of the 8th Mediterranean Conference on Embedded Computing (MECO)*, pages 1–6, 2019.

[14] IEEE-Standard-Association. SHIM 2.0. `https://www.document-center.com/standards/show/IEC-63504-2804`.

[15] David W Walker and Jack J Dongarra. MPI: A Standard Message Pass-

ing Interface. *Journal of Supercomputer*, 12:56–68, 1996.

[16] Ben Meakin and Ganesh Gopalakrishnan. Hardware Design, Synthesis, and Verification of a Multicore Communication API. *Journal of SRC Techcon*, 2(9), 2009.

[17] Shuhei Tsunoda and Takuya Azumi. ROS 2 framework for Embedded Multi-Core Platform. In *Proceedings of the 4th Asia Pacific Conference on Robot IoT System Development and Platform (APRIS)*, 2021.

[18] Michael Cashmore, Maria Fox, Derek Long, Daniele Magazzeni, Bram Ridder, Arnau Carrera, Narcis Palomeras, Natalia Hurtos, and Marc Carreras. Rosplan: Planning in the Robot Operating System. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 333–341, 2015.

[19] Yutaro Kobayashi, Hiroshi Fujimoto, and Takuya Azumi. Communication overhead schema independent of libraries for software/hardware interface. In *Ebook: New Trends in Intelligent Software Methodologies, Tools and Techniques (SOMET)*, pages 30–42, 2022.

[20] Benoît Dupont de Dinechin, Duco van Amstel, Marc Poulhiès, and Guillaume Lager. Time-Critical Computing on a Single-Chip Massively Parallel Processor. In *Proceedings of the 17th IEEE Conference on Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 1–6, 2014.

[21] Hiroyuki Chishiro, Kazutoshi Suito, Tsutomu Ito, Seiya Maeda, Takuya Azumi, Kenji Funaoka, and Shinpei Kato. Towards heterogeneous computing platforms for autonomous driving. In *2019 IEEE International Conference on Embedded Software and Systems (ICESS)*, pages 1–8, 2019.

[22] SiFive. SiFive Performance P870/P870A. `https://www.sifive.com/cores/performance-p870-p870a`.

[23] Zhaoqian Zhong and Masato Edahiro. Model-Based Parallelization for Simulink Models on Multicore CPUs and GPUs. In *Proceedings of the 16th International SoC Design Conference (ISOCC)*, pages 103–104, 2019.

[24] Yamamoto Ryota, Oinuma Masahiro, Kondo Masaki, Honda Shinya, and Edahiro Masato. Multirate Model Parallelization for MPSoC with FPGA in Model-Based Development: A Case Study. In *Proceedings of the 3rd Asia Pacific Conference on Robot IoT System Development and Platform (APRIS)*, pages 6–13, 2020.

[25] Sasuga Kojima, Masato Edahiro, and Takuya Azumi. Remapping Method to Minimize Makespan of Simulink Model for Embedded Multi-Core Systems. In *Proceedings of the 33rd International Conference on Computers and Their Applications (CATA)*, 2018.

[26] Hiro Mikami, Kei Torigoe, Makoto Inokawa, and Masato Edahiro. LLVM Instruction Latency Measurement for Software-Hardware Interface for Multi-many-core. *International Journal of Computer Applications*, 22:50–63, 2022.

[27] Shafiul Azam Chowdhury, Sohil Lal Shrestha, Taylor T Johnson, and Christoph Csallner. SLEMI: Equivalence Modulo Input (EMI) Based Mutation of CPS Models for Finding Compiler Bugs in Simulink. In *Proceedings of the 42nd International Conference on Software Engineering (ICSE)*, pages 335–346, 2020.

[28] Autoware-Foundation. localization_node.hpp. `https://gitlab.com//autoware.auto/AutowareAuto/-/blob/master/src/localization/localization_nodes/include/localization_nodes/localization_node.hpp/`.

Yue Hou is a master's student at the Graduate School of Science and Engineering, Saitama University.

Yutaro Kobayashi is received M.E. degree in computer science from the Graduate School of Science and Engineering, Saitama University in 2023. His research interests include embedded systems and model-based development.

Hiroshi Fujimoto has been a member of Technology Headquarters in eSOL Co., Ltd, since October 2015. He is engaged in the development of parallel processing software that runs on multi/many-cores.

Takuya Azumi is a Professor at the Graduate School of Science and Engineering, Saitama University. He received his Ph.D. degree from the Graduate School of Information Science, Nagoya University. From 2008 to 2010, he was under the research fellowship for young scientists for the Japan Society for the Promotion of Science. From 2010 to 2014, he was an Assistant Professor at the College of Information Science and Engineering, Ritsumeikan University. From 2014 to 2018, he was an Assistant Professor at the Graduate School of Engineering Science, Osaka University. His research interests include real-time operating systems and component-based development. He is a member of IEEE, ACM, IEICE, and JSSST.