Electronic Preprint for Journal of Information Processing Vol.29

Regular Paper

Accurate Contention-aware Scheduling Method on Clustered Many-core Platform

Shingo Igarashi[1,a)] Takuro Fukunaga[2,3,4] Takuya Azumi[1,3]

Received: June 30, 2020, Accepted: December 1, 2020

Abstract: Embedded systems such as self-driving systems require a computing platform with high computing power and low power consumption. Multi-/many-core platforms definitely meet these requirements. However, for hard real-time applications, multiple demands on shared resources can hinder real-time performance. Memory is one of the resources that can most dramatically impair desired performance. Therefore, we addressed contentions induced by shared memory. The ability to predict contentions that may occur during memory access helps to reduce them. We improved the predictability of contentions by dividing tasks into the memory access phase and the execution phase using a Directed Acyclic Graph (DAG). Existing methods can make accurate contention estimations for one Compute Cluster (CC) of a clustered many-core processor. Our method is able to perform accurate contention estimations for multiple CCs, thereby doubling the scalability when contentions are taken into account. Using an Integer Linear Programming (ILP) formulation, we produced a static, non-preemptive, partitioned, and time-triggered schedule. We also conducted an experiment in order to minimize the makespan. The evaluation confirmed that our new method reduced the makespan by increasing the number of CCs.

Keywords: real-time scheduling, many-core, communication contention, directed acyclic graph, integer linear programming

1. Introduction

In recent years, embedded systems such as self-driving systems (e.g., Autoware [1], [2]) demand high computing capacity and low power consumption. Many-core hardware for embedded systems meets these demands and is the object of intense study. Many-core hardware is suitable for parallel tasks processing and for large-scale computations.

Self-driving systems require hard real-time performance. Therefore, the operation and processing need to be determined statically during the development stage. In a self-driving system, multiple applications are running simultaneously and each application has a deadline. Applications such as automatic brakes and collision warnings utilize various types of sensor data and multiple processes. In our paper, we propose a static scheduling method to meet deadlines by accelerating processing using the above-mentioned many-core hardware. Kalray MPPA-256 processor is one of the many-core hardware systems. For the work presented in this paper, we used Kalray MPPA-256 processor.

Cluster-type many-core architectures such as Kalray MPPA-256 have high performance and power efficiency. However, in such systems, resource contentions (such as those induced by shared hardware like shared memory and cache) prevent the system from meeting real-time requirements. In this paper, we solved this problem by estimating effective contentions in order to reduce the makespan. In addition, although there are many cases where accurate contention estimation is performed within one compute cluster (CC) using existing methods [3], [4], [5], we could find no examples of contention-aware scheduling for multiple CCs. (Several schedules have been introduced that are only aware of contentions between CCs [6]). For the purpose of improving scalability and maintaining the real-time functionality, one of the goals of this paper is to extend accurate contention estimates to multiple CCs.

We propose using an ILP (Integer Linear Programming) formulation optimization method, which incorporates awareness of contentions that occur during memory access. A DAG (Directed Acyclic Graph) application is given as input data, and the amount of communication data and the WCET (Worst Case Execution Time) of tasks are statically estimated. This ILP formulation realizes a contention-aware schedule that crosses CCs through NoC communication rather than as a competitive approach within a single CC.

Contributions

- We propose an improved memory access model of tasks for estimating contentions that occurred by assigning tasks to multiple CCs of a clustered multi-/many-core system.
- We propose an optimization method using the ILP formulation that performs DAG scheduling and mapping to minimize the makespan.
- We evaluate the benefits of increasing the number of CCs used and the scalability of the proposed method through

[1] Graduate School of Science and Engineering, Saitama University, Saitama 338–8570, Japan
[2] Faculty of Science and Engineering, Chuo University, Bunkyo, Tokyo 112–8551, Japan
[3] JST, PRESTO, Chiyoda, Tokyo 102–0076, Japan
[4] RIKEN AIP, Cyuo, Tokyo 103–0027, Japan
[a)] s.igarashi.633@ms.saitama-u.ac.jp

© 2021 Information Processing Society of Japan

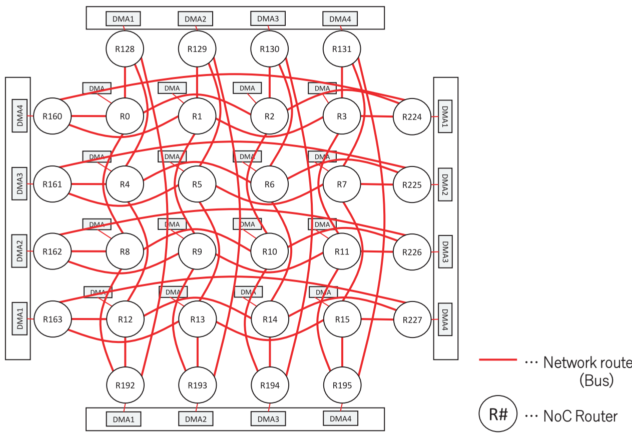

Fig. 1 NoC connections of Kalray MPPA-256.

computational experiments using task graphs converted from multiple benchmarks as input data.

The remainder of this paper is organized as follows. The system model is described in Section 2. Section 3 contains an introduction to the proposed scheduling method for precisely estimating contentions. The scheduling techniques using an ILP formulation are presented in Section 4, and experimental methods, results, and considerations are given in Section 5. Related work is presented in Section 6. Finally, the conclusion and directions for future work are in Section 7.

2. System Model

2.1 Architecture Model

To begin with we will introduce Kalray MPPA-256 processor and its use as many-core hardware for embedded systems [7], [8]. Despite having the 256 + 16 large cores, the power consumption of Kalray MPPA-256 is small. Its power efficiency is 16 W. Therefore, the industry expects Kalray MPPA-256 to be used for embedded systems such as automotive systems. Kalray MPPA-256 has four cores in each I/O cluster labeled north, south, east, and west. Kalray MPPA-256 has 16 CCs in the center, and each CC has 16 cores. Kalray MPPA-256 serves as a means of communication between CCs assembled in a Network on Chip (NoC) formulation. Mutual I/O clusters and CCs constitute a network like the red line in **Fig. 1**. Each CC has 2 MB of SRAM memory shared locally, and 16 cores of each CC access their shared memory through a bus connection [9]. Each I/O cluster contains 2 MB of SRAM and 2 GB of DDR memory. Access from each core to local shared memory is arbitrated by the Round-Robin policy. Data transfer through each memory is realized using DMA. The Resource Manager (RM) in the CC is responsible for managing memory accesses. The RM can copy data from external memory to local memory.

Memory Bank Privatization:

Memory bank privatization is one of the functions of Kalray MPPA-256 necessary for considering contentions [3], [5]. The local memory in the CC is divided into 16 local memory banks, and each of which is assigned to each core as a private memory bank (See **Fig. 2**). Each core has a private access path to its private memory bank, therefore, there is no interference between the core and its private memory bank. Global copies of communica-

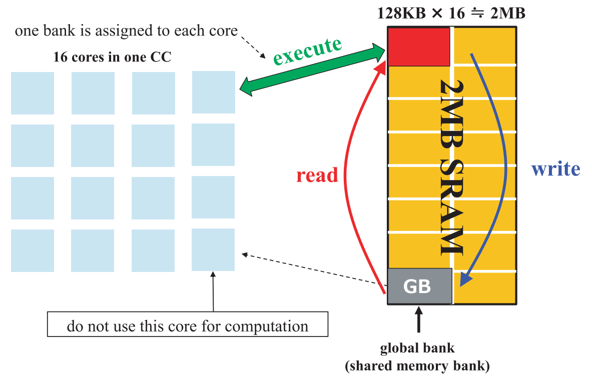

Fig. 2 Memory bank privatization and an example of the read-execute-write semantics in one CC.

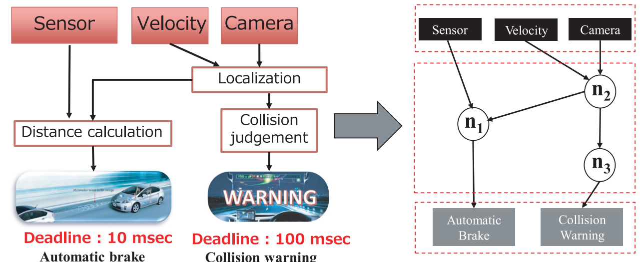

Fig. 3 The structure design of an automotive embedded system (left) and the corresponding DAG representation (right).

tion variables are stored in the *global bank*, and a different bank in CC that can be accessed from all cores as a shared memory for communication. Interference in accessing the global bank is possible. One of the cores in the cluster is called the *management core*, and the remaining cores are compute cores. This management core is RM in Kalray MPPA. The cores assigned to the global bank are not used for computation. The management core manages access to the shared resources from each compute core. For each core, data are transferred between the private memory bank and the global bank through a shared bus. Please note that in this paper, we assume that all the code and data can be covered by 2 MB of SRAM in the CC and do not use off-chip memories such as DDR.

2.2 Application Model

To schedule the numerous applications required in an automotive system, modeling of data flow is necessary. We use a DAG to model such data flow. Applications used in an automotive embedded system utilize stream processing results as shown in the left side of **Fig. 3**. An automotive system can be represented as a DAG by representing each process by a node of the DAG and data flow to or from a process as a directed edge. Each node requires time for computations. Each end node has a deadline. The presence of a directed edge indicates that a pair of nodes has an order constraint or a data dependency.

Read-Execute-Write Semantics:

We adopt the *read-execute-write* semantics found in the papers [3], [4], [5], and an extension of the PREM (Predictable Execution Model) [10], [11], to accurately estimate contentions. Each task (node) of a DAG is divided into three phases: *read, execute*, and *write*. The *read* phase receives and reads the code or data from the global bank and stores it in the core's private

© 2021 Information Processing Society of Japan

memory bank. The *execute* phase can proceed without access to the shared bus. Finally, the *write* phase writes or sends the resulting data to the global bank. The communication phases that access the shared memory via the shared bus or NoC link are the read and write phases. Therefore, we focus on these phases to analyze contentions. By combining *read-execute-write* semantics with the above-mentioned *memory bank privatization*, it is possible to prevent interference during the *execute* phase of each task. An example of the read-execute-write semantics is shown in Fig. 2. In this figure, we visualize the three phases of the task assigned to the computational core in one CC.

Read-execute-write semantics are applicable to the actual application. Autoware is one of the ROS (Robot Operating System)-based open-source self-driving systems and provides the necessary modules for the autonomous vehicles. ROS is middleware used for robot development and open-source software. ROS utilizes a *Publish/Subscribe communication model* as a communication method. In this method, data is called a *topic*, and a program is called a node. *Topics* are pushed by publishing and received by subscribing. ROS handles large-scale processing by separating processes into nodes and communicating between them, thereby enabling distributed processing. We can apply the idea of read-execution-write semantics by considering subscribing as read and publishing as write. This idea has already been introduced as a deterministic scheduling mechanism, called *the rclc Executor*, for ROS 2 applications on microcontrollers [12].

2.3 DAG Notation

In read-execute-write semantics, a task i is denoted by a tuple $\langle \tau_i^r, \tau_i, \tau_i^w \rangle$ representing the read, execute, and write phases (We mostly follow the task notation of the paper [4]). An edge is defined by a tuple $e = \langle \tau_s^w, \tau_t^r, D_{s,t} \rangle$, where τ_s^w is the write phase of the source task s, and τ_t^r is the read phase of the target task t. The quantity of data exchanged between s and t is $D_{s,t}$. When τ_s^w and τ_t^r are mapped to the same core directly using the core's private memory bank, the communication time for the data $D_{s,t}$ is zero because there is no need to access to the global bank.

We use C_i to indicate a WCET. This value is the execution time when the task is executed in a single core architecture. In this situation, the task does not receive interference by other tasks during execution. This is because all required data have been fetched into the private memory bank from the global bank before the task's execution.

On the other hand, in the read phase and the write phase, it is necessary to estimate the delay due to contention as shown in the paper of Rouxel et al. [4] because cores in the CC are connected by the shared bus and memory access can overlap. We extend the method of Rouxel et al. [4] and estimate contentions using NoC communication. Section 3 describes the details.

One objective of this paper is to minimize this makespan. Therefore, the deadline is not specifically set and all the tasks of the DAG are assumed to be the same period. Considering the scheduling extension of multi-rate DAGs (with multiple periods) is a topic for future work. However, dealing with multi-rate DAGs is a separate research topic from contention-aware scheduling, and whether DAGs have the same or multiple periods does not significantly affect this work.

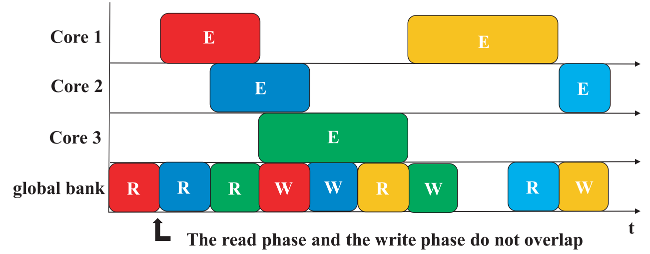

Fig. 4 An example of a time-triggered schedule.

3. Contention-aware Mapping and Scheduling Approach with Two Clusters

In this section, we describe a method for accurately estimating the communication cost. First, we introduce the existing method that applies to a single CC, and then provide the ideas for using the two CCs in the proposed method.

3.1 Estimating Interference in One CC

By introducing a time-triggered schedule, contention can be avoided even in the memory access phase [3], [5]. We therefore will begin the section by explaining a *time-triggered schedule*. As described in Section 2, by combining *Memory Bank privatization* and *read-execute-write* semantics, it is possible to execute the task without contentions, but contention can still occur when tasks access the global bank. As **Fig. 4** shows, a time-triggered schedule coordinates the access to the global bank so that the read phase and the write phase do not overlap.

3.2 The Proposed Contention-aware in 2 CCs

We will extend the existing methods founded in Refs. [3], [4] and propose a contention-aware schedule method targeting two CCs of Kalray MPPA-256. Therefore the problem to solve is more difficult. Like the light blue CC routers in **Fig. 5**, we use two North and South CCs (Note that it is written as CC, not router, for clarity in this figure). It is assumed that two NoC links can be used. One is a link directly connecting the clusters, and the other is a link passing through a router of the I/O cluster (In Fig. 5, R128 and R192). To estimate contentions in two CCs, we consider a time-triggered schedule that accesses to both of the global banks in each CC will not overlap.

To estimate contentions in two CCs, it is necessary to improve the *read phase* mainly in the *read-execute-write* semantics. Let *predecessors*(i) be the set of adjacent preceding tasks on the DAG of task τ_i. Since there is data dependence in the read phase of τ_i, it is necessary to take the variable after the execution of *predecessors*(i) from the global bank of the CCs where the core that executed the task belongs. In this case, NoC communication is performed when at least one of *predecessors*(i) is executed with CC different from τ_i. For example, if τ_i is mapped to the core of CC1 and *predecessors*(i) is executed on the core of CC2, it is necessary to access the global bank of CC2 in the read phase of τ_i, as shown in the left of Fig. 5. At this time, it is necessary to prevent access to the global bank of CC2 from overlapping by a time-triggered schedule as shown in Fig. 4. In the write phase

© 2021 Information Processing Society of Japan

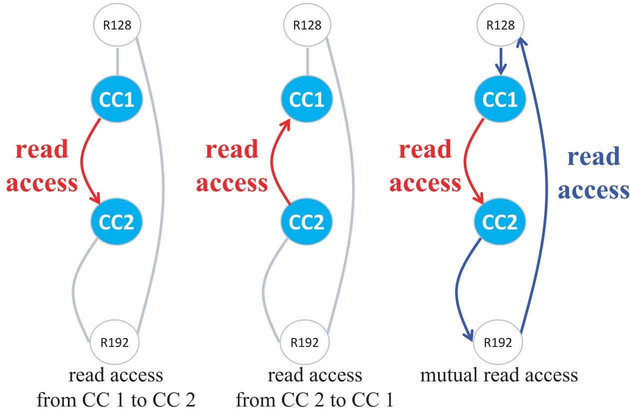

Fig. 5 *Read* phase in two CCs.

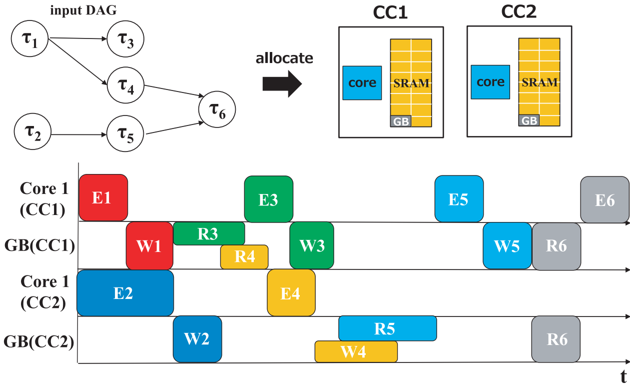

Fig. 6 Schedule example in two CCs.

of the task, data are written to the global bank of the core's CC to which the task is mapped. In other words, no communication via NoC is performed in the write phase. This is because if the task mapping and the memory to be written are not the same CC, detection of dependency data of the preceding task during the read phase becomes difficult, and it is not scalable. With this setting, it is possible to decide which global bank should be accessed in the read phase.

An example of a proposed schedule is visualized in **Fig. 6**. We assume that the DAG at the upper left of Fig. 6 is scheduled to be allocated to two CCs each of which has one compute core. The expected schedule is similar to the one at the bottom of Fig. 6. In Fig. 6, paying attention to the execution of τ_3, since the preceding τ_1 is executed on core 1 of the CC1 and τ_3 is also mapped to CC1, then the global bank (GB) of CC1 is accessed in the read phase of τ_3. On the other hand, noting τ_4, since the preceding τ_1 is executed in CC1, and τ_4 is mapped to core1 of CC2, in the read phase of τ_4, only the GB of CC1 is accessed. Finally, focusing on τ_6, τ_6 is mapped to CC1, preceding task τ_4 was executed by CC2, and preceding task τ_5 is executed by CC1. Therefore, in the read phase of τ_6, GBs of both CC1 and CC2 are accessed. Note that read access to the two GBs is done at the same time as the read phase of τ_6 in Fig. 6. We assume that during the read phase, data obtained via NoC communication is directly placed in the core's private memory bank, not the GB.

We allow overlapping of communication phases because we are aiming to minimize makespan (Contention-free schedule is not necessarily the shortest schedule [4]). In other words, it is better to allow partial overlap if it is effective in minimizing the makespan even if contentions are not completely avoided. Therefore, it is necessary to find the shortest schedule by calculating the latency due to contention.

The communication cost of a communication phase (read and write) depends on how much interference occurs. The Interference is due to tasks running in parallel on different cores because access to the global bank is arbitrated by a Round-Robin policy through the shared bus in the CC. A pair of tasks is concurrent if they do not have dependencies between each other and may be run in parallel. For example, in Fig. 6, tasks τ_3 and τ_4 are concurrent. To accurately estimate contentions, it is necessary to recognize concurrent tasks assigned to other cores and count the number of tasks with the overlap in access to the same memory [4]. Depending on the number of tasks to overlap, the execution time of the communication phase changes and affects the makespan.

Although NoC communications may be used during the read phase of a task, these communications have one of the following three patterns as shown in Fig. 5. Since there are two CCs and two NoC links to be used, by creating a time-triggered schedule, no contention occurs even in communication through a NoC. Therefore, it is not necessary to describe constraint equations that limit NoC links. Furthermore, this paper does not consider the memory capacity, so it does not include variables that indicate memory allocation.

4. Proposed ILP Method

In this section, we formulate the scheduling method described in Section 3.2 with ILP. We schedule and map the task graph on the multi-/many-core platform for the purpose of getting the shortest schedule. **Table 1** summarizes the notations and variables needed for the ILP formulation.

For a concise presentation of the constraints, the logical operators $\wedge$ and $\vee$ are used in place of the words 'and' and 'or', respectively. These operators can be transformed into linear constraints using the simple transformation rules from Ref. [13], as shown in **Table 2**.

Objective function. The goal of the optimization is to minimize the makespan of the schedule, which is the latest completion time of the scheduled tasks, as shown in Eq. (1). Let T be a set of tasks. Equation (2) gives the constraints on completion time of each task, namely that the start time of the write phase, ρ_i^w, plus its WCET, $delay_i^w$, be less than or equal to the makespan of the schedule.

$$minimize\ \Theta \tag{1}$$

$$\forall i \in T; \rho_i^w + delay_i^w \leq \Theta \tag{2}$$

Variables for specifying allocations. Some basic rules of scheduling are shown in the following equations. Equation (3) guarantees that one task is mapped to a single CC. A set of CCs is denoted by CC and the binary variable $q_{i,cc}$ returns 1 if task τ_i is assigned to the core of CC cc. $sameCC_{i,j}$ indicates whether two tasks are mapped to the same CC. Equation (4) demands the $sameCC_{i,j}$ returns 1 if tasks τ_i and τ_j are assigned to different cores of the same CC. Similarly, Eq. (5) guarantees that a task is

© 2021 Information Processing Society of Japan

Table 1 Notations and ILP variables.

Sets	
T	the set of tasks
CC	the set of compute cluster
P	the set of processors/cores
Functions	
$predecessors(i)$	returns the set of adjacent predecessors of task τ_i
$successors(i)$	returns the set of adjacent successors of task τ_i
$are_conc(i)$	returns the set of concurrent tasks of with task τ_i
Constants	
C_i	task τ_i *execute* phase's WCET (Worst Case Execution Time) computed in isolation
$D_{i,j}$	amount of data exchanged between task τ_i and τ_j
$\mathcal{M}$	big-M constant used for activating or withdrawing a constraint
Integer variables	
$\rho_i^r, \rho_i^e, \rho_i^w$	start times of *read*, *execute*, and *write* phases of task τ_i
δ_i^r, δ_i^w	total amount of data read/written by a *read/write* phase of τ_i according to predecessors/successors' mapping
$chunk_i^r, chunk_i^w$	the number of full slots to *read/write*
$remainingTime_i^r, remainingTime_i^w$	the remaining time to a *read/write* phase
$waitingSlots_i^r, waitingSlots_i^w$	the number of full slots a *read/write* phase
$interf_i^r, interf_i^w$	the number of interfering tasks of the *read/write* phases of τ_i
$delay_i^r, delay_i^w$	task τ_i *read/write* phase's WCET
$unused_rest_i^r, unused_rest_i^w$	the remaining allocation time of the duration T_{slot}
Binary variables	
$p_{i,c} = 1$	task τ_i is mapped on core c
$q_{i,cc} = 1$	task τ_i is mapped on CC cc
$m_{i,j} = 1$	tasks τ_i & τ_j are mapped on core c
$sameCC_{i,j} = 1$	tasks τ_i & τ_j are mapped on the same compute cluster
$a_{i,j} = 1$	task τ_i is scheduled before task τ_j, in the sense $\rho_i^r \leq \rho_j^r$
$am_{i,j} = 1$	same as $a_{i,j}$ but on the same core
$HereCC_i = 1$	all $predecessors(i)$ are executed on the same CC as task τ_i
$AnotherCC_i = 1$	all $predecessors(i)$ are executed on the CC different from task τ_i
$TwoCC_i = 1$	$predecessors(i)$ are executed on both CCs
$ov_{i,j}^{XY}$	phase X of τ_i overlaps with phase Y of τ_j, $XY \in \{rr, ww, rw, wr\}$ (r : *read* phase, w : *write* phase)
$CCov_{i,j}^{XY}$	phase X of τ_i overlaps with phase Y of τ_j when at least one phase crosses CC, $XY \in \{rr, rw, wr\}$
Objective function	
Θ	the makespan of the schedule

Table 2 Convert to linear equations.

$c = a \wedge b$	$c + 1 \geq a + b$	$c \leq a$	$c \leq b$
$c = a \vee b$	$c \leq a + b$	$c \geq a$	$c \geq b$

mapped to a single core. The set of processors or cores is denoted by P and the binary variable $p_{i,c}$ returns 1 if task τ_i is assigned to the processor or core c. Equation (6) indicates whether two tasks are mapped to the same core. The binary variable $m_{i,j}$ returns 1 if tasks τ_i and τ_j are assigned to the same core, while the variable $a_{i,j}$ imposes the order of two tasks τ_i and τ_j, namely it indicates that task τ_i is scheduled before task τ_j if $a_{i,j} = 1$. Equation (7) forces at least one of $a_{i,j}$ and $a_{j,i}$ to be 1. Since variable $a_{i,j}$ is used for task ordering on the same core, variables $a_{i,j}$ and $a_{j,i}$ can be restricted as in Eq. (7). Equation (8) defines the binary variable $am_{i,j}$ returns 1 if task τ_i is scheduled before task τ_j on the same core.

$$\forall i \in T;\ \sum_{cc \in CC} q_{i,cc} = 1 \tag{3}$$

$$\forall (i, j) \in T \times T; i \neq j;$$
$$sameCC_{i,j} = \sum_{cc \in CC} (q_{i,cc} \wedge q_{j,cc}) \text{ and } sameCC_{i,j} = sameCC_{j,i} \tag{4}$$

$$\forall i \in T; \sum_{c \in P} p_{i,c} = 1 \tag{5}$$

$$\forall (i, j) \in T \times T; i \neq j;$$
$$m_{i,j} = \sum_{c \in P} (p_{i,c} \wedge p_{j,c}) \text{ and } m_{i,j} = m_{j,i} \tag{6}$$

$$\forall (i, j) \in T \times T; i \neq j;\ a_{i,j} + a_{j,i} = 1 \tag{7}$$

$$\forall (i, j) \in T \times T; i \neq j;\ am_{i,j} = a_{i,j} \wedge m_{i,j} \tag{8}$$

Read-execute-write semantics constraints. Each phase must be executed contiguously. The start time of the execute phase of task τ_i (ρ_i^e) immediately follows the completion of the read phase (where the start of the read phase is ρ_i^r + communication cost $delay_i^r$) as shown in Eq. (9). Similarly, the write phase (ρ_i^w) begins immediately after the completion of the execute phase (start of the execute phase ρ_i^e + WCET C_i), as shown in Eq. (10). If the phases are non-consecutive, the problem will be greatly complicated. Therefore, this paper imposes these constraints.

$$\forall i \in T;\ \rho_i^e = \rho_i^r + delay_i^r \tag{9}$$

$$\forall i \in T;\ \rho_i^w = \rho_i^e + C_i \tag{10}$$

Tasks on the same core do not overlap. Since each core cannot process more than one task in parallel, we require a constraint that $rho_i^w + delay_i^w <= rho_j^r$ for two tasks τ_i and τ_j, if τ_i and τ_j are mapped to the same core and τ_i is scheduled before τ_j (i.e., $am_{i,j} = 1$). On the other hand, this constraint should not be applied otherwise. To formulate this situation as a linear constraint, we use the big-M method [14]. Let $\mathcal{M}$ be a large constant that satisfies

$$\mathcal{M} = \sum_{i \in T} (C_i + delay_i^r + delay_i^w) \tag{11}$$

Then, we have a constraint

© 2021 Information Processing Society of Japan

$$\forall(i,j) \in T \times T; i \neq j;\ \rho_i^w + delay_i^w \leq \rho_j^r + \mathcal{M} \cdot (1 - am_{i,j}) \quad (12)$$

Because of Eq. (11), $\mathcal{M}$ is larger than or equal to the left-hand side of Eq. (12). Hence, if $am_{i,j} = 0$, then constraint Eq. (12) is satisfied irrespective of rho_i^w, $delay_i^w$, and rho_j^r. If $am_{i,j} = 1$, then the right-hand side of Eq. (12) is rho_i^r, and thus it represents the required constraint.

Data dependencies in the task graph. Equation (13) enforces data dependencies by constraining all tasks to start after the completion of all their respective predecessors. Let $predecessors(i)$ denotes the set of tasks preceding τ_i in the task graph.

$$\forall i \in T, \forall j \in predecessors(i);\ \rho_j^w + delay_j^w \leq \rho_i^r \quad (13)$$

Determining communication phases overlap. For each pair of communication phases, we introduce variables that indicate if the phases are overlapping in the schedule ($ov_{i,j}^{XY} = 1$, with $X \in \{rr, ww, rw, wr\}$ - r : *read* phase, w : *write* phase). This variable is true when phase X of task τ_i and phase Y of task τ_j, where memory access is performed, overlap in time. This variable can be used to determine if a memory contention is occurring, and we can calculate the communication delay. In addition, interference with tasks located in the other cluster outside the cluster can occur. At this time, communication using NoC is performed. In order to distinguish between communication in one CC and NoC communication, we set different variables. Therefore, we use a variable not found in the existing method [4], namely $ov_{i,j}^{XY}$, to determine the overlap of the communication phases of the tasks in a single CC. The overlap when NoC communication occurs is expressed as the variables $CCov_{i,j}^{XY}$.

First, we introduce variables to distinguish the various cases of overlap between communication phases. The following three variables determine how to map the adjacent preceding tasks to CCs for each task. $sameCC_{i,j}$ indicates whether two tasks are mapped to the same CC. When $HereCC_i$ is 1, all the tasks that directly precede task τ_i are executed with the same CC as task τ_i. The total power function in the formula actually means a continuation of logical conjunction ($\wedge$). According to Ref. [13], the constraint equation is expressed on the code using a plurality of expressions without using multiplication as shown in Table 2. Conversely, $AnotherCC_i$ becomes 1 if all the tasks that directly precede task τ_i were executed by a CC different from that of task τ_i. Finally, if neither $HereCC_i$ nor $AnotherCC_i$ are 1, then a prior task was executed in each CC and the value of $TwoCC_i$ is 1.

$$\forall i \in T;$$

$$HereCC_i = \begin{cases} 1 & \text{if} \prod_{k \in predecessors(i)} sameCC_{i,k} = 1 \\ 0 & \text{otherwise} \end{cases} \quad (14)$$

$$AnotherCC_i = \begin{cases} 1 & \text{if} \prod_{k \in predecessors(i)} \neg sameCC_{i,k} = 1 \\ 0 & \text{otherwise} \end{cases} \quad (15)$$

$$TwoCC_i = \begin{cases} 1 & \text{if } HereCC_i = 0 \wedge AnotherCC_i = 0 \\ 0 & \text{otherwise} \end{cases} \quad (16)$$

Using the three variables defined above, it is possible to create variables to judge the overlap between concurrent tasks. If task τ_i and task τ_j are concurrent, one can use the $are_conc(i)$ function to find the concurrent task τ_j given task τ_i. The function $are_conc(i)$ builds the transitive closure of the DAG and judges a pair of tasks that are not connected by edges in the graph as concurrent [4]. For overlap judgment, it is necessary to check whether the communication phase of a task overlaps with the communication phase of its concurrent task, and whether NoC communication is being performed in the read phase. All constraint equations are visualized in **Fig. 7**, showing only one example of the many possible cases for each variable.

(1) rr : Overlap between *read* phases. Equation (17) makes a judgment of an overlap between read phases within a single CC, that is, it identifies the case where tasks τ_i and τ_j and all the preceding tasks are all mapped to the same CC. δ_i^r indicates the quantity of data to be read.

$$\forall i \in T, \forall j \in are_conc(i);$$

$$ov_{i,j}^{rr} = HereCC_i \wedge HereCC_j \wedge sameCC_{i,j} \wedge \delta_i^r > 0 \wedge \delta_j^r > 0 \wedge \rho_i^r < \rho_j^e \wedge \rho_j^r < \rho_i^e \quad (17)$$

The following four patterns must be considered for the overlap between two read phases between two CCs. Equation (18) considers the case of accessing the global bank of a CC different from the node to which task τ_j is assigned during the read phase of task τ_j. Equation (19) is the opposite case of Eq. (18). Equation (20) considers the case where both read phases overlap when accessing the CC in which the tasks are not placed. Equation (21) considers the case where at least one of the read phases of τ_i and task τ_j accesses the global bank of both CCs, that is, the case where the preceding task was executed in both CCs.

$$\forall i \in T, \forall j \in are_conc(i),$$

$$CC_A ov_{i,j}^{rr} = HereCC_i \wedge AnotherCC_j \wedge \neg sameCC_{i,j} \wedge \delta_i^r > 0 \wedge \delta_j^r > 0 \wedge \rho_i^r < \rho_j^e \wedge \rho_j^r < \rho_i^e \quad (18)$$

$$CC_B ov_{i,j}^{rr} = AnotherCC_i \wedge HereCC_j \wedge \neg sameCC_{i,j} \wedge \delta_i^r > 0 \wedge \delta_j^r > 0 \wedge \rho_i^r < \rho_j^e \wedge \rho_j^r < \rho_i^e \quad (19)$$

$$CC_C ov_{i,j}^{rr} = AnotherCC_i \wedge AnotherCC_j \wedge sameCC_{i,j} \wedge \delta_i^r > 0 \wedge \delta_j^r > 0 \wedge \rho_i^r < \rho_j^e \wedge \rho_j^r < \rho_i^e \quad (20)$$

$$CC_D ov_{i,j}^{rr} = TwoCC_i \vee TwoCC_j \wedge \delta_i^r > 0 \wedge \delta_j^r > 0 \wedge \rho_i^r < \rho_j^e \wedge \rho_j^r < \rho_i^e \quad (21)$$

Since only one equation from Eqs. (18) to (21) is actually adopted in any given situation, these can be connected by a logical or ($\vee$) as in the following Eq. (22).

$$CCov_{i,j}^{rr} = CC_A ov_{i,j}^{rr} \vee CC_B ov_{i,j}^{rr} \vee CC_C ov_{i,j}^{rr} \vee CC_D ov_{i,j}^{rr} \quad (22)$$

(2) ww : Overlap between *write* phases. Since NoC is not used in the write phase (as mentioned in Section 3), the overlap between write phases can be represented by Eq. (23), given below.

$$\forall i \in T, \forall j \in are_conc(i),$$

$$ov_{i,j}^{ww} = sameCC_{i,j} \wedge \delta_i^w > 0 \wedge \delta_j^w > 0 \wedge \rho_i^w < \rho_j^w + delay_j^w \wedge \rho_j^w < \rho_i^w + delay_i^w \quad (23)$$

(3) rw : Overlap of *read* phase and *write* phase. One must be

© 2021 Information Processing Society of Japan

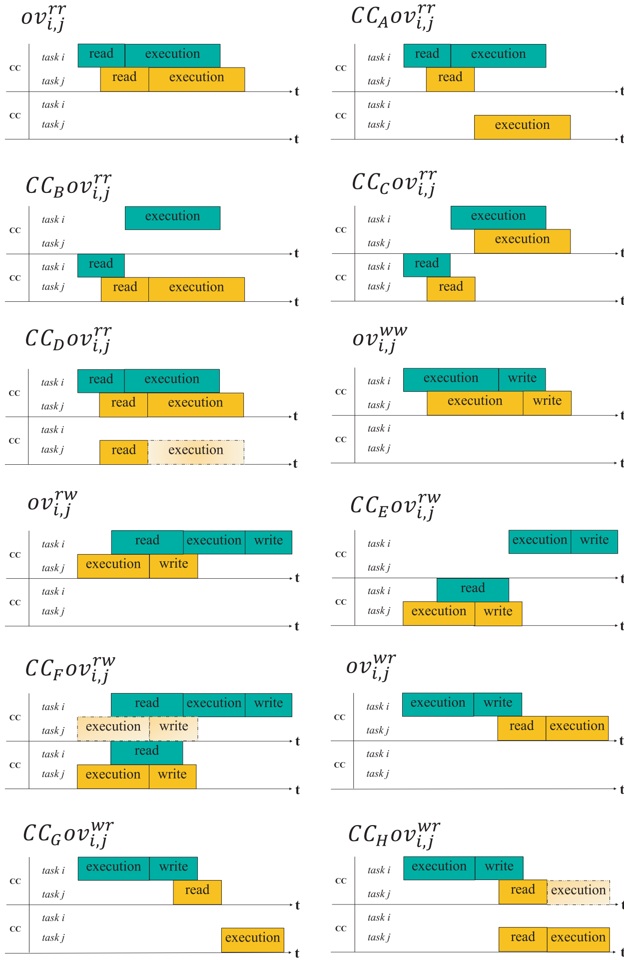

Fig. 7 An example of each overlapping variable.

aware that there is a possibility of NoC communication only in the read phase of task τ_i, because the write phase writes data to only the core where that task is mapped. Therefore, unlike *rr*, only the task mapping of the adjacent predecessor of task τ_i is used for the determination. Equation (24) determines the overlap of the read phase and the write phase of two tasks mapped in the same CC. Equation (25) considers a case in which only a CC to which task τ_i does not belong to is accessed in the read phase of τ_i. Equation (26) considers the case where $predecessors(i)$ were executed in both CCs.

$$\forall i \in T, \forall j \in are_conc(i);$$

$$\begin{aligned} ov^{rw}_{i,j} = {} & HereCC_i \land sameCC_{i,j} \\ & \land\ \delta^r_i > 0 \ \land\ \delta^w_j > 0 \\ & \land\ \rho^r_i < \rho^w_j + delay^w_j \land \rho^w_j < \rho^e_i \end{aligned} \tag{24}$$

$$\begin{aligned} CC_Eov^{rw}_{i,j} = {} & AnotherCC_i \land \neg sameCC_{i,j} \\ & \land\ \delta^r_i > 0 \ \land\ \delta^w_j > 0 \\ & \land\ \rho^r_i < \rho^w_j + delay^w_j \land \rho^w_j < \rho^e_i \end{aligned} \tag{25}$$

$$\begin{aligned} CC_Fov^{rw}_{i,j} = {} & TwoCC_i \\ & \land\ \delta^r_i > 0 \ \land\ \delta^w_j > 0 \\ & \land\ \rho^r_i < \rho^w_j + delay^w_j \land \rho^w_j < \rho^e_i \end{aligned} \tag{26}$$

(4) *wr* : Overlap of *write* phase and *read* phase. For the same reason as *rw*, the Eqs. (27), (28), (29), given below, can be derived. In the case of *rw*, task's phases were reversed, and the necessary explanation was in *rw*, so a detailed explanation will be omitted.

$$\forall i \in T, \forall j \in are_conc(i);$$

$$\begin{aligned} ov^{wr}_{i,j} = {} & HereCC_j \land sameCC_{i,j} \\ & \land\ \delta^w_i > 0 \ \land\ \delta^r_j > 0 \\ & \land\ \rho^w_i < \rho^e_j \land \rho^r_j < \rho^w_i + delay^w_i \end{aligned} \tag{27}$$

$$\begin{aligned} CC_Gov^{wr}_{i,j} = {} & AnotherCC_j \land \neg sameCC_{i,j} \\ & \land\ \delta^w_i > 0 \ \land\ \delta^r_j > 0 \\ & \land\ \rho^w_i < \rho^e_j \land \rho^r_j < \rho^w_i + delay^w_i \end{aligned} \tag{28}$$

$$\begin{aligned} CC_Hov^{wr}_{i,j} = {} & TwoCC_j \\ & \land\ \delta^w_i > 0 \ \land\ \delta^r_j > 0 \\ & \land\ \rho^w_i < \rho^e_j \land \rho^r_j < \rho^w_i + delay^w_i \end{aligned} \tag{29}$$

Equations (30) and (31) estimate the number of instances of interference for each of the read phase and write phase of the task. Variable $interf^r_i$ and $interf^w_i$ count the number of instances of interference during the read phase and the write phase of task i respectively. The number of instances of interference cannot be larger than the number of cores. Note that for overlaps other than *ww*, only either *ov* or *CCov* is adopted.

$$\forall i \in T;$$

$$\begin{aligned} interf^r_i = \sum_{\forall p \in P} \Big[\bigvee_{\forall j \in T | are_conc(i)} & (ov^{rr}_{i,j} \lor CCov^{rr}_{i,j} \lor ov^{rw}_{i,j} \\ & \lor\ CC_Eov^{rw}_{i,j} \lor CC_Fov^{rw}_{i,j}) \land p_{j,p} \Big] \end{aligned} \tag{30}$$

$$\begin{aligned} interf^w_i = \sum_{\forall p \in P} \Big[\bigvee_{\forall j \in T | are_conc(i)} & (ov^{ww}_{i,j} \lor ov^{wr}_{i,j} \lor CC_Gov^{wr}_{i,j} \\ & \lor CC_Hov^{wr}_{i,j}) \land p_{j,p} \Big] \end{aligned} \tag{31}$$

Finally, to improve the solving time, Eq. (32) contains optimizations that constrain the overlapping variables.

$$\forall i \in T, \forall j \in are_conc(i);$$

$$\begin{aligned} ov^{rr}_{i,j} &= ov^{rr}_{j,i} & CC_Aov^{rr}_{i,j} &= CC_Bov^{rr}_{j,i} \\ CC_Cov^{rr}_{i,j} &= CC_Cov^{rr}_{j,i} & CC_Dov^{rr}_{i,j} &= CC_Dov^{rr}_{j,i} \\ ov^{ww}_{i,j} &= ov^{ww}_{j,i} & ov^{wr}_{i,j} &= ov^{rw}_{j,i} \\ CC_Gov^{wr}_{i,j} &= CC_Eov^{rw}_{j,i} & CC_Hov^{wr}_{i,j} &= CC_Fov^{rw}_{j,i} \end{aligned} \tag{32}$$

Estimation of the duration of worst-case communication. To estimate the length of the communication phase, one needs the quantity of data to be read and written (δ^r_i, δ^w_i). In addition, when two consecutive tasks are mapped to the same core (i.e., $m_{i,j} = 1$), communication overlap does not occur. This leads to Eqs. (33) and (34) below. The function $successors(i)$ denotes the set of adjacent successors of task i. The constant $D_{i,j}$ is the amount of data transmitted from task τ_i to task τ_j. The variable δ is the sum of

© 2021 Information Processing Society of Japan

the quantities of data to be communicated during each communication phase excluding the quantity of communication on the same core as the task.

$$\forall i \in T;$$
$$\delta_i^r = \sum_{j \in predecessors(i)} D_{j,i} \cdot (1 - m_{j,i}) \quad (33)$$
$$\delta_i^w = \sum_{j \in successors(i)} D_{i,j} \cdot (1 - m_{i,j}) \quad (34)$$

The following four expressions encode the Round-Robin bus arbitration policy [4]. Maximum duration, T_{slot}, is allocated to each core of each CC. During T_{slot}, up to D_{slot} data words can be transferred as in Eq. (35) (i.e., one data word is transferred at T_{slot}/D_{slot} time units). When there is more data than D_{slot} can accommodate, it is divided and transmitted as *chunks* as in Eq. (35). One *chunk* is transmitted in T_{slot}, and the remaining data are transmitted in *remainingTime* (see Eq. (36), $remainingTime < T_{slot}$). The access time of the core to the memory is represented by the second and subsequent items of Eq. (38). Each chunk is kept waiting at a multiple of T_{slot} according to the number of overlapping concurrent tasks. Therefore, if *waitingslot* is defined to be large as in Eq. (37), the waiting time can be estimated as in the first term of Eq. (38). We define the total delay time in Eq. (38) by adding the waiting time and the access time. Equations for the write phase are completely analogous (i.e., r - w), and, therefore, are omitted.

$$\forall i \in T;$$
$$chunks_i^r = \left\lfloor \frac{\delta_i^r}{D_{slot}} \right\rfloor \quad (35)$$
$$remainingTime_i^r = (\delta_i^r \bmod D_{slot}) \cdot \left(\frac{T_{slot}}{D_{slot}} \right) \quad (36)$$
$$waitingSlots_i^r = \left\lceil \frac{\delta_i^r}{D_{slot}} \right\rceil \quad (37)$$
$$delay_i^r = T_{slot} \cdot waitingSlots_i^r \cdot interf_i^r + T_{slot} \cdot chunks_i^r + remainingTime_i^r \quad (38)$$

Since Eqs. (35)–(37) are not linear, they are linearized and are expressed as follows. Variable $unused_rest_i^r$ means the remaining allocation time, that is, the duration $T_{slot} - remainingTime_i^r$.

$$\forall i \in T;$$
$$\delta_i^r = chunks_i^r \cdot T_{slot} + remainingTime_i^r \quad (39)$$
$$\delta_i^r = waitingSlots_i^r \cdot T_{slot} - unused_rest_i^r \quad (40)$$
$$unused_rest_i^r \geq remainingTime_i^r \quad (41)$$

Equation (38) is quadratic and non-convex. We solve this problem using a safe linear approximation of Eq. (38). Following the method of Rouxel et al. [4], we replaced $waitingSlots_i^r$ with $WAIT_i^r$. For the worst case, we let this variable overestimate the number of $waitingSlot_i^r$. Therefore, $WAIT_r^i$ is defined as the sum of all data read $D_{j,i}$ as in Eq. (42).

$$\forall i \in T; \ WAIT_i^r = \left\lceil \left(\sum_{j \in predecessors(i)} D_{j,i} \right) \Big/ T_{slot} \right\rceil \quad (42)$$

Table 3 StreamIt benchmark suite scheduling.

Name	#Tasks	Width	avg data	avg WCET (time unit)
Audiobeam	20	15	12 B	41
Beamformer	56	12	18 B	2718
BitonicSort	122	8	49 B	30
DCTcomp	13	3	768 B	4557
DCTverif	7	2	513 B	10045
FFT2	26	2	551 B	618
FFT3	82	16	84 B	120
FFT4	10	2	6 B	11
FFT5	115	16	52 B	38
Firbank	340	12	505 B	670
FMRadio	67	20	6 B	235
FilterbankNew	53	8	35 B	144
MatrixMultiBlock	23	2	793 B	726
Serpent	234	2	1013 B	922
dcalc	84	4	106 B	174
perftest	16	4	8267 B	5269

For bus communication and NoC communication, the length of memory access differs by about a factor of three with a simple value [3]; however, it cannot be considered in this paper. This consideration of the difference in communication time complicates the optimization problem. In addition, since it is difficult to set the speed of NoC communication with an accurate value, it is difficult to express a strict speed difference as an ILP. On the other hand, in the case of heuristic approaches, it is easy to see the tendency due to the change in the communication speed value by changing the communication speed parameter. Therefore, this is left as a topic for future work.

5. Evaluations

5.1 Evaluation Method

As input data, we used applications from the StreamIT benchmark suite modeled as fork-join graphs [4], [15], [16]. **Table 3** summarizes the benchmarks used in our experiments and the number and width of tasks (the maximum number of tasks that can be executed in parallel), the average of the quantity of data communicated between tasks, and the average of the WCETs of each task.

We adopted the contention-aware approach of Rouxel et al. [4] for one CC as a comparative method in which a schedule is generated assuming that there are two cores in one CC. The proposed method uses two CCs with two cores per one CC (four cores in total). It is impossible to ascertain the benefits of the proposed method simply by increasing the number of clusters; therefore, we used methods with a different number of cores in our comparison and set $Tslot = 3$.

In the evaluations, we used CPLEX [17] version 12.8.0 as an ILP solver. We ran CPLEX using up to 32 threads of an Intel Xeon Gold 6148 processor (2.4 GHz, 20 cores, 40 threads) in the AI Bridging Cloud Infrastructure. The upper limit of the continuous login time to the computing node of the cloud is 12 hours; therefore, we used CPLEX with a timeout of 11.5 hours.

5.2 Comparison Result with Existing Method

We compared the schedule makespan obtained and solving time of the two methods. The result of the evaluation is summarized in **Table 4**, which shows the makespan by each method, the solving time, whether the solution is optimal, and the number

© 2021 Information Processing Society of Japan

Table 4 Evaluation results.

Name	one CC [4]			two CCs (proposed)			gain[%]
	makespan	time[s]	variables	makespan	time[s]	variables	
Audiobeam	4561^f	timeout	3722	4034^f	timeout	6285	11.554
Beamformer	101832^f	timeout	24990	49406^f	timeout	40329	51.482
BitonicSort	196624^f	timeout	59694	-	timeout	106733	
DCTcomp	135454°	14.09	1005	119457°	1.24	1630	11.810
DCTverif	119212°	0.06	144	119212°	0.12	312	0.000
FFT2	3986530°	10.89	5005	3982430°	14.29	8519	0.001
FFT4	7272°	0.28	192	6753°	0.74	521	0.071
FFT3	452498°	4992.13	34709	-	timeout	60545	
FFT5	131688^f	timeout	69799	-	timeout	122678	
Firbank	-	timeout	968014	-	timeout	2603543	
FMRadio	-	timeout	33126	-	timeout	52746	
FilterbankNew	128626°	4569.93	32774	-	timeout	54183	
MatrixMultiBlock	301564°	16206.50	2760	220540°	11.61	4848	26.87
Serpent	3977670°	9793.6	163948	3977670°	402.782	301831	0.000
dcalc	152704^f	timeout	25578	125072^f	timeout	46045	18.95
perftest	2157190°	151.06	1957	2002570°	24.178	3562	7.13

o : optimal solution

f : feasible solution

- : no solution

of ILP variables. The gain of the table is calculated using each makespan as Eq. (43).

$$gain = \frac{oneCC - twoCCs}{oneCC} \times 100 \quad (43)$$

From the value of gain, our method succeeds in getting a shorter makespan. The input graphs whose makespan has not changed (*DCTverif* and *Serpent*) seem to have not improved even if the number of clusters was increased because the width is two (i.e., the optimum solution is founded at the time of the existing method). On the contrary, there are graphs in which width is 2, but the makespan becomes short, because wait time in the communication phase is reduced as the number of memory has increased. In other words, even if the same width occurs, whether the makespan becomes shorter depends on the structure of the graph in addition to the number of clusters. From the above analysis on the makespan, we believe that we could create a scheduling approach using two CCs considering contentions.

From the comparison of the solving time values, our ILP formulation could solve the problem quickly while seeking a better makespan. However, because the time required for estimation increases dramatically with the number of tasks, width, and the number of cores, it does not always produce a solution in a reasonable time. From the execution result of the input graph *Audiobeam*, it turns out that in particular the parallelism degree greatly influences the solving time. We believe this is due to the existence of concurrent tasks and all the possible overlapping cases. Therefore, there were some graphs that could not find any solution within the time limit. In the proposed method, the number of cores is doubled, and the number of variables related to core mapping increased. Therefore, the number of ILP variables is approximately doubled. On the other hand, the increase in the number of variables by the proposed method for identifying clusters is only a small percentage as a whole. Since the number of cores to be used also affects both the schedule time and the solving time, it is necessary to use a heuristic approach for graphs that could not obtain a solution this time.

5.3 Evaluation of the Scalability of the Proposed Method

In this subsection, we use the proposed method and observe the improvement in makespan when the number of cores is changed. We selected two applications, *Audiobeam* and *Beamformer* of

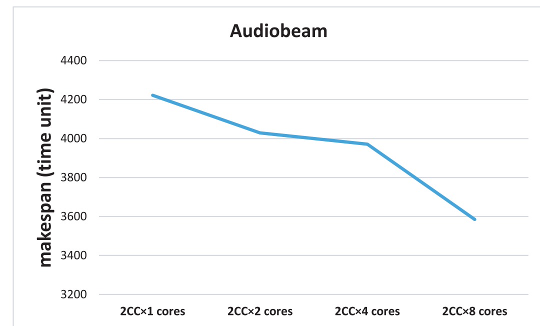

Fig. 8 Improvement by increasing the number of cores (Audiobeam).

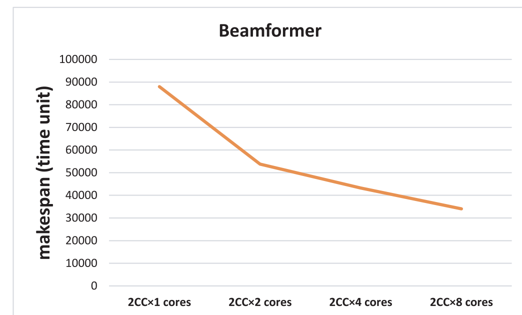

Fig. 9 Improvement by increasing the number of cores (Beamformer).

the benchmark applications. These were chosen because of their high width (parallelism) and the fact that we could find solutions in Table 4. In this evaluation, we ran CPLEX using an Intel Core i7-8550U CPU (1.8 GHz, four cores, and eight threads), and the CPLEX timeout was set to 24 hours.

The result of the evaluation is summarized in **Fig. 8** and **Fig. 9**. All the solutions here were feasible. It can be confirmed that the makespan of each application is significantly reduced as the number of cores used increases. In addition, since the proposed method accurately estimates the delay due to contention, we think that it is suitable for hard real-time systems such as self-driving systems, which require a larger amount of calculation than the existing method.

The proposed method is applicable to two clusters and is capable of running independent applications on eight pairs of two clusters, taking into account delays due to contention. This means that the proposed method can utilize all 16 clusters of Kalray MPPA2-256 Bostan. However, the time to generate a schedule will become impractical as the size and number of applications grow because the proposed method is an optimal approach.

To make full use of high scalability of the many-core, we need to create a fast heuristic approach using equivalent ideas. When such work is completed, the ILP method presented here will provide a baseline against which to confirm the consistency of the heuristic method.

The proposed method is a theoretical scheduling approach, and it is impossible to compare it with the competitive delay in actual hardware. The allocation time for each request depends on the hardware, and in a multi-/many-core processor, many contentions can occur. Therefore, it is very difficult to estimate the contention between specific tasks when the application is actually running.

© 2021 Information Processing Society of Japan

Table 5 Comparison with previous work.

	our paper	A [5]	B [3]	C [4]	D [6]
DAG scheduling	✓			✓	
contention-aware in one CC	✓	✓	✓	✓	
contention-aware between CCs with NoC communication	✓				✓
ILP (CP) formulation	✓	✓	✓	✓	✓

A : contention-free framework of Becker et al. [5]

B : contention-free scheduling using a CP formulation of Becker et al. [3]

C : contention-aware scheduling of Rouxel et al. [4]

D : NoC partitioned method of Becker et al. [6]

6. Related Work

Contention analysis of shared resources in multi-/many-core platforms has received significant attention in recent years. Part of contention analysis is task scheduling. Task scheduling on multi-/many-core platforms consists of deciding where (mapping) and when (scheduling) each task is executed [18], [19], [20], [21], [22], [23], [24]. **Table 5** summarizes a comparison of this paper with its main related research.

Becker et al. [5] proposed an ILP formulation and a heuristic for scheduling periodic, independent, PREM-based tasks on one cluster of Kalray MPPA processor. They systematically created a contention-free schedule and proposed an execution framework for avoiding contention by taking advantage of memory privatization features available in processors such as Kalray MPPA-256. They divided tasks into 3 sub-tasks: *read* copies input data from the shared memory to a private memory bank, *execute* only accesses the private memory bank, and *write* copies output data to the shared memory. Using a specific scheduling policy, it is possible to avoid contention completely. Becker et al. [3] extended their own approach in Ref. [5] and improved scalability by distinguishing task types and efficiently using both off-chip memory and local memory. In the case of the scheduling method for automotive applications, it is essential to use off-chip memory (DDR); therefore, we used a different task model. However, the approach of Becker et al. [3] should be examined in the future.

Rouxel et al. [4] introduced two contention-aware schedule strategies that generate a time-triggered schedule for application tasks. They accurately estimated contentions and established the ILP formulation of the scheduling problem and a heuristic approach. The proposed technique reduced the makespan compared to the case of the worst contentions. On the basis of their success, we proposed a method to generate a schedule using two CC. Becker et al. [5] considered sporadic independent tasks for the purpose of finding a valid schedule to satisfy the deadline of each task, while Rouxel et al. [4] sought to find the shortest schedule by considering one iteration of a task graph.

Becker et al. [6] proposed a new partition strategy for Kalray MPPA-256 to reduce contentions on NoC. They divided 16 CCs into two groups with north and south, to ensure each CC has a private link to the north or south off-chip memory (DDR).

7. Conclusion

This paper presents a proposed method for estimating contentions when tasks access shared memory using two CCs of Kalray MPPA-256 processor. By improving the memory access model, the proposed method divides the task into an execute phase and the communication phase, and can estimate contentions in both phases within a single CC and between CCs. We incorporated this idea as an optimization method using the ILP formulation, and performed DAG scheduling and mapping to find a schedule that minimized contentions. Input data was modeled by using StreamIT benchmark suite as a DAG. Our evaluation results showed that, compared to the existing method, our approach improved the schedule makespan, confirming that the processing time of the application can be reduced by increasing the number of CCs used. However, since the proposed method calculates all interference patterns, the solving time required to solve the optimization problem greatly increases with an increase in the number of tasks, parallelism, and the number of cores.

Future work is to refine the contention model and make it more usable for many-core for CC platforms such as Kalray MPPA-256 processor by limiting contentions and guaranteeing real-time performance. Furthermore, it is necessary to create an ILP model in consideration of the memory capacity and our memory model. In addition, we will examine a high-speed heuristic method expected to be equivalent to the proposed method, and consider a multi-rate (multiple periods) DAG. The heuristic method will also be able to model the difference in communication speed both inside and outside the cluster.

Acknowledgments We thank Benjamin Rouxel, University of Rennes, for sharing the experimental environment and answering our questions. This work was partially supported JST PRESTO Grant Number JPMJPR1751 and JPMJPR1759.

References

[1] autoware's Official Website: Autoware.AI, available from ⟨https://www.autoware.ai/⟩.

[2] Kato, S., Tokunaga, S., Maruyama, Y., Maeda, S., Hirabayashi, M., Kitsukawa, Y., Monrroy, A., Ando, T., Fujii, Y. and Azumi, T.: Autoware on board: Enabling autonomous vehicles with embedded systems, *Proc. ICCPS*, pp.287–296 (2018).

[3] Becker, M., Mubeen, S., Dasari, D., Behnam, M. and Nolte, T.: Scheduling multi-rate real-time applications on clustered many-core architectures with memory constraints, *Proc. ASP-DAC*, pp.560–567 (2018).

[4] Rouxel, B., Derrien, S. and Puaut, I.: Tightening contention delays while scheduling parallel applications on multi-core architectures, *ACM Trans. Embedded Computing Systems* (*TECS*), pp.164:1–164:20 (2017).

[5] Becker, M., Dasari, D., Nicolic, B., Akesson, B., Nélis, V. and Nolte, T.: Contention-free execution of automotive applications on a clustered many-core platform, *Proc. ECRTS*, pp.14–24 (2016).

[6] Becker, M., Nikolic, B., Dasari, D., Akesson, B., Nélis, V., Behnam, M. and Nolte, T.: Partitioning and analysis of the network-on-chip on a COTS many-core platform, *Proc. RTAS*, pp.101–112 (2017).

[7] Maruyama, Y., Kato, S. and Azumi, T.: Exploring scalable data allocation and parallel computing on NoC-based embedded many cores, *Proc. ICCD*, pp.225–228 (2017).

[8] official Website of Kalray MPPA®: Kalray MPPA® architecture, available from ⟨https://www.kalrayinc.com/portfolio/processors/⟩.

[9] De Dinechin, B.D., Van Amstel, D., Poulhiès, M. and Lager, G.: Time-critical computing on a single-chip massively parallel processor, *Proc. DATE*, p.97 (2014).

[10] Pellizzoni, R., Betti, E., Bak, S., Yao, G., Criswell, J., Caccamo, M. and Kegley, R.: A predictable execution model for cots-based embedded systems, *Proc. RTAS*, pp.269–279 (2011).

[11] Alhammad, A. and Pellizzoni, R.: Time-predictable execution of multithreaded applications on multicore systems, *Proc. DATE*, pp.1–6 (2014).

[12] Real-Time Executor: micro-ROS, available from ⟨https://micro-ros.github.io/docs/concepts/client_library/real-time_executor/⟩.

[13] Brown, G.G. and Dell, R.F.: Formulating integer linear programs: A

© 2021 Information Processing Society of Japan

rogues' gallery, *INFORMS Trans. Education*, Vol.7, No.2, pp.153–159 (2007).
[14] Griva, I., Nash, S.G. and Sofer, A.: Linear and nonlinear optimization, second edition, *Society for Industrial Mathematics* (2008).
[15] Rouxel, B. and Puaut, I.: STR2RTS: Refactored StreamIT benchmarks into statically analysable parallel benchmarks for WCET estimation & real-time scheduling, *Proc. OASIcs-OpenAccess Series in Informatics*, Vol.57, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2017).
[16] Rouxel, B.: Minimising communication costs impact when scheduling real-time applications on multi-core architectures (2018) (online), available from ⟨https://hal.inria.fr/tel-01945456⟩.
[17] official Website of CPLEX: ILOG CPLEX Optimization Studio – Overview, available from ⟨https://www.ibm.com/products/ilog-cplex-optimization-studio⟩.
[18] Nguyen, V.A., Hardy, D. and Puaut, I.: Cache-conscious offline real-time task scheduling for multi-core processors, *Proc. ECRTS* (2017).
[19] Martinez, S., Hardy, D. and Puaut, I.: Quantifying WCET reduction of parallel applications by introducing slack time to limit resource contention, *Proc. RTAS* pp.188–197 (2017).
[20] Igarashi, S., Kitagawa, Y., Ishigooka, T., Horiguchi, T. and Azumi, T.: Multi-rate DAG scheduling considering communication contention for NoC-based embedded many-core processor, *Proc. DS-RT* (2019).
[21] Perret, Q., Maurere, P., Noulard, E., Pagetti, C., Sainrat, P. and Triquet, B.: Temporal isolation of hard real-time applications on many-core processors, *Proc. RTAS*, pp.1–11 (2016).
[22] Perret, Q., Maurère, P., Noulard, É., Pagetti, C., Sainrat, P. and Triquet, B.: Mapping hard real-time applications on many-core processors, *Proc. RTNS*, pp.235–244 (2016).
[23] Rouxel, B., Skalistis, S., Derrien, S. and Puaut, I.: Hiding communication delays in contention-free execution for SPM-based multi-core architectures, *Proc. ECRTS*, pp.1–24 (2019).
[24] Kitagawa, Y., Ishigooka, T. and Azumi, T.: DAG scheduling algorithm for a cluster-based many-core architecture, pp.150–157, DOI: 10.1109/EUC.2018.00030 (2018).

Shingo Igarashi is a master student of Graduate School of Science and Engineering, Saitama University. He received his B.E. degree from School of Science and Engineering, Saitama University in 2019. His research interests include embedded systems, and real-time scheduling.

Takuro Fukunaga is an Associate Professor at Faculty of Science and Engineering, Chuo University. He received the Ph.D. degree in Informatics from Kyoto University, Japan in 2007. He was an Assistant Professor at Kyoto University from 2007 to 2013, a Project Associate Professor at National Institute of Informatics from 2013 to 2017, and a research scientist at RIKEN Center for Advanced Intelligence Project from 2017 to 2019. His research interests include design and analysis of discrete algorithms for combinatorial optimization problems, and their applications to related areas such as operations research, computer communication, and machine learning. He is a member of Operations Research Society of Japan.

Takuya Azumi is an Associate 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 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.

© 2021 Information Processing Society of Japan