PAPER

A Fine-Grained Multicasting of Configuration Data for Coarse-Grained Reconfigurable Architectures

Takuya KOJIMA[†a)], *Student Member and* Hideharu AMANO[†], *Fellow*

SUMMARY A novel configuration data compression technique for coarse-grained reconfigurable architectures (CGRAs) is proposed. Reducing the size of configuration data of CGRAs shortens the reconfiguration time especially when the communication bandwidth between a CGRA and a host CPU is limited. In addition, it saves energy consumption of configuration cache and controller. The proposed technique is based on a multicast configuration technique called RoMultiC, which reduces the configuration time by multicasting the same data to multiple PEs (Processing Elements) with two bit-maps. Scheduling algorithms for an optimizing the order of multicasting have been proposed. However, the multicasting is possible only if each PE has completely the same configuration. In general, configuration data for CGRAs can be divided into some fields like machine code formats of general perpose CPUs. The proposed scheme confines a part of fields for multicasting so that the possibility of multicasting more PEs can be increased. This paper analyzes algorithms to find a configuration pattern which maximizes the number of multicasted PEs. We implemented the proposed scheme to CMA (Cool Mega Array), a straight forward CGRA as a case study. Experimental results show that the proposed method achieves 40.0% smaller configuration than a previous method for an image processing application at maximum. The exploration of the multicasted grain size reveals the effective grain size for each algorithm. Furthermore, since both a dynamic power consumption of the configuration controller and a configuration time are improved, it achieves 50.1% reduction of the energy consumption for the configuration with a negligible area overhead.

key words: *CGRA, configuration reduction, integer-linear-program, multicasting*

1. Introduction

Coarse-grained reconfigurable architecture (CGRA) is a remarkable platform for embedded systems including IoT devices because of its high degree of energy efficiency and programmability. CGRAs have an array of reconfigurable processing elements (PEs) for efficient parallel execution of compute-intensive application. Each PE operates according to configuration data which specify behaviors of components in the PE, and the size of configuration data for the whole PE array is often quite large.

Considering that a CGRA is connected to a host CPU with a general purpose interface bus like 32-bit width as an accelerator, a critical issue is a long time to transfer the configuration data from the CPU to the CGRA, in other words, reconfiguration time. Most of CGRAs are supposed to be used in task-by-task off loading for improving the performance and energy efficiency. Since the energy efficiency of the CGRA is much better than that of the host processor, it is advantageous to off-load tasks as possible. By shortoning the configuration time, the oppotunity to off-load tasks is increased, because small granularity tasks can be a candidate of off-loading. Thus, various researches have be exerted to reduce the configuration time of CGRAs [2], [6], [7].

Similar researches for FPGAs took an approach using bit-level compression techniques such as LZSS, Huffman coding, and run-length-encoding. However, these compressors and decompressors are complex and sometimes require a large buffer which consumes a large energy and semiconductor area. Thus, they are difficult to be used when the energy efficiency is important. Besides, these techniques require that the configuration data has many common bit patterns and the size of configuration data must be enough large to pay for the large compression overhead. Hence, they are suitable for FPGAs [12]–[14] but not for the CGRAs.

Multicasting is one of techniques for reducing the reconfiguration time with simple hardware. It enables the multiple PEs to be configured simultaneously by exploiting a fact that a certain number of PEs use the same configuration data. The multicasting is effective, especially when the same data-flow-graph mapping is repeated on the PE array to utilize data-level parallelism of an application.

RoMultiC is a multicasting method using two bit-maps for the rows and the columns of the PE array [1], [2]. If the configuration data are allowed to be overwritten, the latest configuration for each PE is actually used for its operation. A scheduling of multicasting proposed in [1] can reduce the time for multicasting.

In general, the configuration data can be divided into some fields such as an opcode and operands for ALU. In RoMultiC [1], the PEs which have an identical configuration for all fields are allowed to be multicasted. However, the grain size of multicasted fields has not been well investigated. When the partial fields are allowed to be written, the number of PEs to be multicasted is increased. Here, we propose a fine grain multicasting method considering a practical configuration data format. In this work, we compare two scheduling algorithms: 1) *Espresso* [3]-base algorithm and 2) Integer-Linear-Program (ILP)-base algorithm for the fine grain multicasting. Next, we carry out further exploration for the grain size based on a statistical analysis. Thereby, an effective grain size for each algorithm is revealed.

Implementation overhead is also important as well as reducing the configuration time. Thus, the proposed schemes are implemented considering a real chip CGRA,

Manuscript received October 4, 2018.
Manuscript revised January 11, 2019.
Manuscript publicized April 5, 2019.

[†]The authors are with Dept. of Information and Computer Science, Keio University, Yokohama-shi, 223–8522 Japan.
a) E-mail: wasmii@am.ics.keio.ac.jp

DOI: 10.1587/transinf.2018EDP7336

Copyright © 2019 The Institute of Electronics, Information and Communication Engineers

and compared with a double buffer method from the viewpoint of hardware overhead.

The contributions of this paper are as follows.

- A fine grain multicasting scheme is proposed for further reduction of reconfiguration time,
- we analyze a trade-off between two algorithms for finding multicasting bitmaps,
- the grain size exploration is performed, and
- we show the proposed method is more energy efficient than other methods.

The rest of the paper is organized as follows. The background of this paper and related work are introduced in Sect. 2. Then, Sect. 3 and Sect. 4 give basic concept and algorithms of the fine grain multicasting followed by experimental results in Sect. 5. Finally, the conclusion of this paper is summarized in Sect. 6.

2. Background

2.1 Base Architecture

CMA (Cool Mega Array) is a low power straight forward style CGRA [4]. In this work, we use CC-SOTB (CMA-Cube-SOTB) [5], which is an improved version of CMA, as a base architecture for implementation of the proposed method.

Figure 1 illustrates an array of 12×8 PEs in the CC-SOTB. Each PE consists of an ALU and a switch element (SE) as shown in Fig. 1. PEs are connected each other with two types of interconnections: 1) direct links and 2) links provided by SEs. The CC-SOTB uses a single-context configuration. It can change tasks by transdering another configuration for the next task. Thereby, a large power consumption for a dynamic reconfiguration which changes the configuration cycle-by-cycle can be cut down.

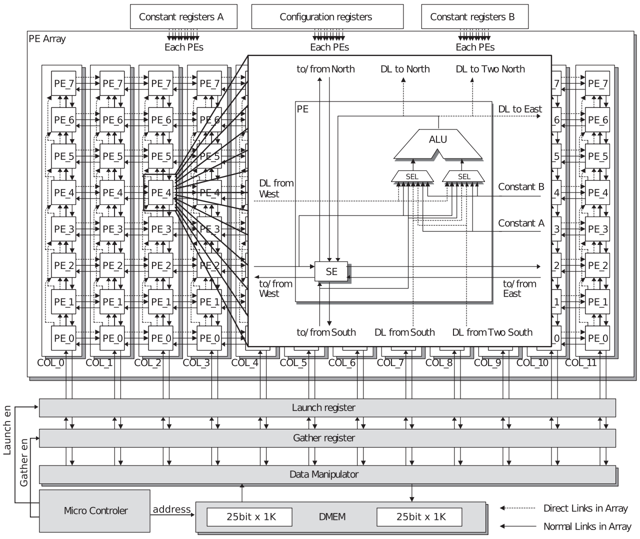


Fig. 1 Overview of the PE array

CC-SOTB is composed of several modules as illustrated in Fig. 2. The micro-controller controls data transfer between the PE array and data memory using micro-instructions stored in instruction memory. The constant registers provide constant values to the PE array. The configuration controller decodes input configuration data and writes them into the configuration registers. CC-SOTB has an external bus and a chip interface for connections to a host CPU or other accelerators. The external bus consists of a 22bit address bus and a 32bit data bus. Each module is mapped to the same address space as shown in Fig. 3. Thereby, the host CPU can access data in CC-SOTB modules via the interface and the external bus. Data for each module are 4-Byte aligned regardless of actual data width for compatibility with the host CPU which we assume is a 32bit processor. Data width of the PE is 25bit so that the size of 2K words data memory is totally 2048 × 25 = 50 Mbit. However, it

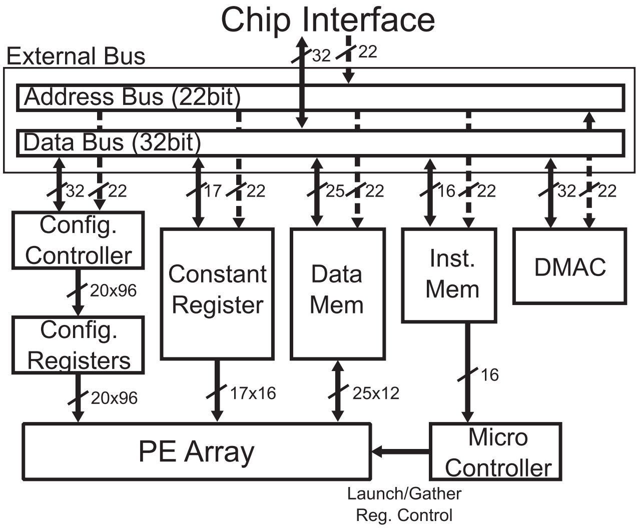


Fig. 2 Architecture of CC-SOTB

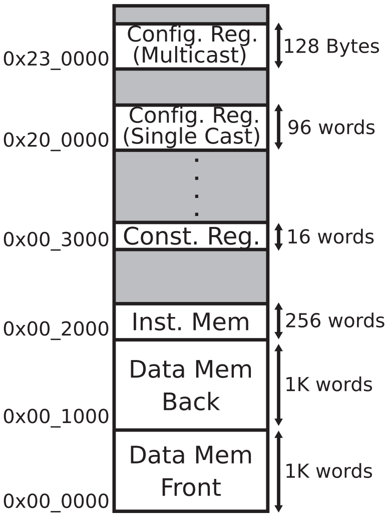


Fig. 3 Address map of CC-SOTB

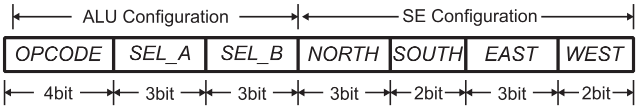


Fig. 4 Configuration data for CC-SOTB

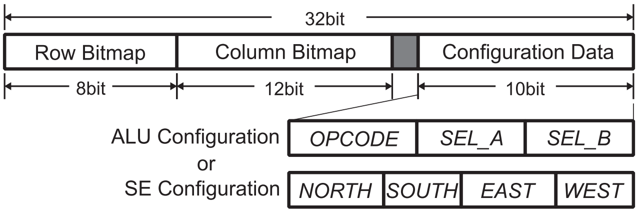


Fig. 5 Multicasting of Configuration data

occupies 2048×32 = 64 Mbit address space. The data memory is separated into two blocks (front and back) for double buffering.

2.2 Configuration Data Format

The configuration data for a PE are described in Fig. 4. *OPCODE, SEL_A*, and *SEL_B* respectively are used for the operation of ALU and the selections of two multiplexers for ALU inputs. *NORTH, SOUTH, EAST, and WEST* are used to specify the routing in an SE. 20-bit per a PE is needed so that the total amount of the configuration data is 20-bit × 96 = 1920-bit. A configuration of each PE is also mapped to the address space. Therefore, it takes 96 cycles to complete the reconfiguration without multicasting.

2.3 Multicast Configuration

In order to reduce the reconfiguration time, RoMultiC scheme [1], which is one of multicasting method, is applied to the CC-SOTB. In RoMultiC, the transferred data use two bit-maps which respectively indicate multicasted rows and columns in the PE array. As for a PE at the coordinate (x, y), the configuration data are multicasted when both x-th bit of the column bit-map and y-th bit of the row bit-map are set to "1". The data format for multicasting is shown in Fig. 5. Unlike an original format used in [1], fields of the configuration data are divided into 2 parts: 1) ALU part and 2) SE part. The bit-maps need 8 + 12 = 20 bits and then 12 bits remain for the configuration data of PEs. The part shaded gray (2 bit) in Fig. 5 is unused.

In order to show how overwriting reduces the configuration data, we use a simple example illustrated in Fig. 6. The example illustrates that the multicasting with the overwrite takes 3 cycles to complete the target configuration. First, configuration "*A*" is multicasted to the whole PEs. Next, configuration "*B*" is multicasted to two PEs at the coordinates (1, 2) and (2, 2). Lastly, configuration "*C*" is multicasted to two PEs at the coordinates (3, 2) and (3, 3). Without the overwrite, it takes 4 cycles since the configuration "*A*"s are completed with multicasting twice.

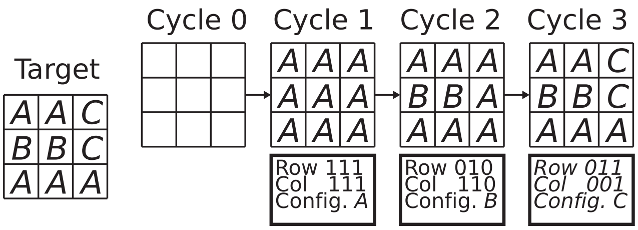


Fig. 6 Efficiency of overwriting configuration

2.4 Related Work

In general, configuration data of CGRAs have two types of localities: 1) temporal locality and 2) spatial locality. With the temporal locality, a PE keeps the same configuration during a few cycles. If the spatial locality exists, the same configuration in multiple PEs can be found at the same time. This work focuses on single-context configuration CGRAs so that only the spatial locality is considered. Likewise, other techniques for reducing the configuration data exploit the spatial locality with multicasting [2], [6], [7]. A hybrid approach which combines multicasting with LZSS compression is proposed in [2]. In methods used in [6] and [7], the range of multicasting is limited in order to minimize an area overhead. Each of them does not aim to improve the spatial locality while this work improves it by changing the grain of multicasting. Also, our paper [8] at the earlier stage of this research only proposes the concept of the fine grain multicasting. It does not demonstrate its effectiveness in a real application. In addition, the grain size exploration of multicasting is not performed in [8].

The dynamic reconfiguration is applied to some CGRAs for high performance computing. In such a CGRA, the temporal locality is also utilized. The compression technique in [9] reduces the configuration data by eliminating the configuration unchanged from the previous cycle. It considers a group of fields which are compressed together depending on a mapped application as well. Due to the complexity of the compression, a genetic algorithm and an integer linear program (ILP) are used. A dictionary-based compression is proposed in [10]. This method separates two types of dictionaries based on the locality of a mapped application. Thereby, even if small dictionaries are employed, enough compression ratio is achieved. [11] classifies each field of the configuration data into three groups: necessary fields, optional fields, and unnecessary fields. Then, the latter two fields are compressed adaptively.

In FPGAs (Field-Programmable Gate Arrays), configuration data reduction is also important so that many techniques are proposed. FPGAs usually have larger configuration data than CGRAs. If the configuration data is enough large, the same bits pattern could appear many times. Therefore, dictionary-based compression algorithms such as LZSS are often used for FPGAs. A technique in [12] focuses not only on a compression ratio (CR) but also on a decompression efficiency. It uses a dictionary-based compression with bitmasks, which is originally proposed in [13]. In addition, run-length-encoding is employed in order to achieve high CR. The bitmasks also help enhance the CR. Even though data which try to be compressed does not match any entries of the dictionary, it can be compressed with the bitmask. When there is only a few bit difference between the data and an entry, the bitmask is used to specify the difference. Besides, [12] proposes a scheme to convert a variable length coding into a fixed length coding. It enables the decompressor to be more simple and high throughput. [14] takes an approach using LZ77. According to its analyses, there is variation in matching length. Therefore, the method employs Golomb coding for the matching length. [15] carries out trade-off analysis among resource utilization, CR and decompression throughput. [16] utilizes Golomb coding for both smaller compression time and smaller compressed data than run-length-encoding.

In summary, most of the methods for FPGAs exploit bit-level common patterns. However, as mentioned above, CGRA's configuration data is relatively small, especially when the CGRA follows the task-by-task reconfiguration policy. Therefore, for CGRAs, the bit-level compression is not expected to make a significant contribution on the compression ratio, or rather causes a large hardware overhead.

3. Motivation

3.1 Fine Grain Multicasting

As explained in Sect. 2.2, the configuration data are multicasted ALU-by-ALU or SE-by-SE. It is not clear that the grain of multicasting is really effective in reducing the configuration data. Here, we consider to use smaller units and propose the fine grain multicasting.

In order to implement the scheme to CC-SOTB, a new data format for the multicasting is described in Fig. 7. In the fine grain multicasting, any combination of the fields can be multicasted unless the required data size exceeds in the available data size. Optimized combination of the fields can increase the possibility of either multicasting more PEs or utilizing the available data space in the format. Figure 7 shows an example of the best case that the data space is fully utilized. In this case, the configuration data include 4 fields which require 12 bits totally.

Although additional flag bits are necessary to decide which fields the configuration data contain, in our case, it can be included in the address space. In other words, only 128-byte address space has to be reserved for the fine grain multicasting. Assuming n fields, 2^n-byte address space is needed. In this work, we reserve an address space from 0x23_0000 to 0x23_007F as described in Fig. 3. The lowest 7bits of the address are used for the flag bits. Of course, the flag bits are included in the data format. Nevertheless, it increases the overhead for multicasting in addition to the bit-maps.

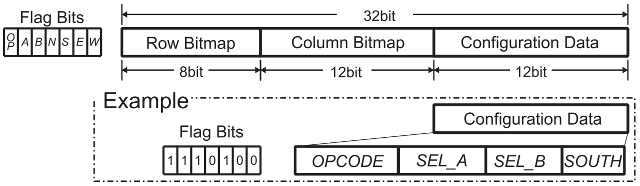


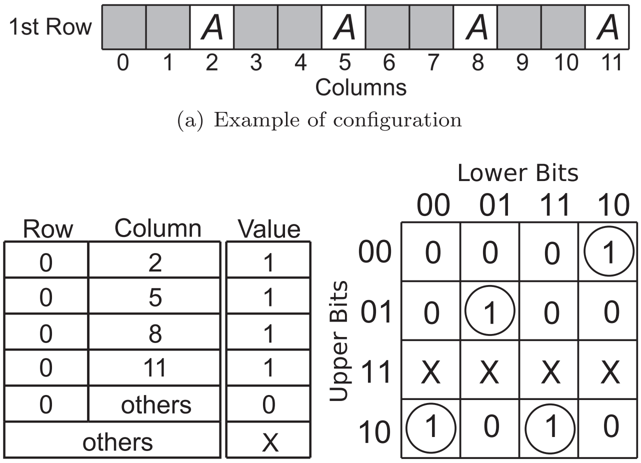
Fig. 7 Fine grain multicasting

3.2 Using *Espresso*

As a way to find the suitable bit-maps, a natural way is using CAD algorithms. Here, *Espresso*, a classic logic minimization algorithm, is employed like [17]. It can treat "Don't Care" denoted by **X** as well as logic **0** and **1**. A truth table whose entry corresponds to each PE in the PE array is used in *Espresso*. **1** indicates that the PE has the same configuration data, while **0** means that the PE has a different configuration data and is already written. **X** is used when a PE has a different configuration data but have not been written yet.

Although *Espresso* itself is a highly efficient heuristic algorithm, sometimes it does not work well to find the bit-maps. For instance, when a target configuration shown in Fig. 8 (a) is given, a truth table in Fig. 8 (b) for multicasting configuration "*A*" is generated. The parts shaded gray indicate PEs whose configuration is already written. Then, a Karnaugh's map associated with the row bit-map is created as described in Fig. 8 (c). However, a cube which covers multiple cells cannot be found. A cube corresponds to a group of multicasted PEs. In other words, the four PEs which have the configuration "A" cannot be multicasted simultaneously.

It should be discussed how the disadvantage of *Espresso* for the fine grain multicasting has an influence on the reduction of configuration data. Here, we propose another method using Integer-Linear-Program in the next section.



(a) Example of configuration

Row	Column	Value
0	2	1
0	5	1
0	8	1
0	11	1
0	others	0
others		X

(b) Truth Table

Upper Bits \ Lower Bits	00	01	11	10
00	0	0	0	1
01	0	1	0	0
11	X	X	X	X
10	1	0	1	0

(c) Karnaugh map

Fig. 8 Worst case with *espresso*

Algorithm 1 Greedy Algorithm for Overwrite Scheduling

```
Input: $C$
Output: $C_{ordered}$
 1: $C_{fix} \leftarrow \phi, C_{unfix} \leftarrow C$
 2: while $C_{unfix} \neq \phi$ do
 3:     $B_r, B_c, f, d \leftarrow$ find_maximize_pattern($C_{fix}, C_{unfix}$)
 4:     $M \leftarrow$ multicasted_set($B_r, B_c, f, d$)
 5:     $C_{fix} \leftarrow C_{fix} + M$
 6:     $C_{unfix} \leftarrow C_{unfix} - M$
 7:     insert($C_{ordered}, M$)
 8: end while
```

4. New Multicasting Method

4.1 Scheduling Algorithm

As mentioned in Sect. 2.3, scheduling an order of the overwrite is important to reduce the configuration data. Here, we employ a greedy algorithm for scheduling similarly to [1], [17] because of the simplicity of the algorithm.

The scheduling algorithm is described in Algorithm 1. Given a target configuration C, finding the bit-maps which maximize the number of written bits is repeated until the configurations of all fields in the PEs are fixed. Please note that the bits of multicasted fields which are overwritten later are not counted for maximizing.

In this work, two algorithms are presented as the function "find_maximize_pattern" respectively in the Sect. 4.2 and the Sect. 4.3. Both algorithms return bit-maps for the rows B_r and the columns B_c, the group of fields in the configuration format f, and multicasted data d. The returned values are inserted into an ordered configuration data C_{order}.

4.2 *Espresso* for Finding the Bit-Maps

Algorithm 2 shows a solution for finding the bit-maps with *Espresso*. C_{fix} is a set of fixed configurations and they cannot be overwritten except when all multicasted fields are the same as the one previously written. In this case, the value of the truth table is set to **X** (Don't care).

Like Fig. 8, a truth table tt for each combination of the fields f and for each configuration data d are generated. 2^F denotes the power set of the fields F, that is, possible combinations of the fields. In our case, $|F|$ equals 7 so that $\left|2^F\right|$ equals 128. However, there exist invalid combinations because some of the combinations require more than available data space (12 bit). Therefore, the number of the valid combinations is 94. The validation of the combinations is checked at line. 4 in Algorithm 2.

After *Espresso* finds cubes, bit-maps of the rows and the columns for each cube are generated. Then, the sum of written bits is calculated and the best bit-maps $B_{r,max}$ and $B_{c,max}$ are obtained. This method does not maximize the number of bits multicasted at the same time, but maximizes the number of multicasted rows and columns for given combination of the fields. This is because *Espresso* can treat only a single output logic.

Algorithm 2 Algorithm for obtaining bitmaps with *Espresso*

```
Input: $C_{fix}, C_{unfix}$
Output: $B_{max,r}, B_{max,c}, f_{max}, d_{max}$
 1: $S_{max} \leftarrow 0, B_{r,max} \leftarrow$ None, $B_{c,max} \leftarrow$ None,
    $f_{max} \leftarrow \phi, d_{max} \leftarrow$ None
 2: $F = \{OPCODE, SEL\_A, SEL\_B,$
         $NORTH, SOUTH, EAST, WEST\}$
 3: for all $f \in 2^F$ do
 4:   if bit_width($f$) $\leq$ max_width then
 5:     $\boldsymbol{D} \leftarrow$ enumerate_config_data($C_{unfix}, f$)
 6:     for all $d \in \boldsymbol{D}$ do
 7:       $tt$ = make_truth_table($C_{fix}, C_{unfix}, d, f$)
 8:       $\boldsymbol{B_r}, \boldsymbol{B_c} = espresso(tt)$
 9:       for all $b_r, b_c \in \boldsymbol{B_r}, \boldsymbol{B_c}$ do
10:         $S \leftarrow$ count_bits($b_r, b_c, f$)
11:         if $S > S_{max}$ then
12:           $S_{max} \leftarrow S$, $B_{r,max} \leftarrow b_r$, $B_{c,max} \leftarrow b_c$
13:           $f_{max} \leftarrow f$, $d_{max} \leftarrow d$
14:         end if
15:       end for
16:     end for
17:   end if
18: end for
```

Algorithm 3 Algorithm for obtaining bitmaps with ILP

```
Input: $C_{fix}, C_{unfix}$
Output: $B_{max,r}, B_{max,c}, f_{max}, d_{max}$
 1: $S_{max} \leftarrow 0$
 2: for all $d \in$ enumerate_config_data($C_{unfix}$) do
 3:   $S, B_r, B_c, f \leftarrow$ find_bitmap_by_ILP($C_{fix}, C_{unfix}, d$)
 4:   if $S_{max} > S$ then
 5:     $S_{max} \leftarrow S, B_{max,r} \leftarrow B_r, B_{max,c} \leftarrow B_c$
 6:     $f_{max} \leftarrow f, d_{max} \leftarrow d$
 7:   end if
 8: end for
```

4.3 ILP for Finding the Bit-Maps

In addition to the *Espresso*-based algorithm, we consider using an integer linear program (ILP) in order to obtain an optimal solution of the bit-maps. An overview of the algorithm is shown in Algorithm 3. In addition, an ILP model used in the function "find_bitmap_by_ILP" is formulated as follows.

$$isField_i = \begin{cases} 1 & \text{if the } i\text{-th field} \\ & \quad \text{is written} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$isRow_j = \begin{cases} 1 & \text{if the } j\text{-th row} \\ & \quad \text{is written} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

$$isCol_k = \begin{cases} 1 & \text{if the } k\text{-th column} \\ & \quad \text{is written} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

$$\max S = \sum_i \sum_j \sum_k S_{ijk} * isField_i * isRow_j * isCol_k \tag{4}$$

subject to

$$\sum_{i} bit\_width_i * isField_i \le bit\_width_{max} \tag{5}$$

if i-th field of the PE in the j-th row and the k-th column is already fixed, then

$$isField_i = 0 \lor isRow_j = 0 \lor isCol_k = 0 \tag{6}$$

where S_{ijk} is the number of bits for i-th field of the PE in the j-th row and the k-th column, bit_width_i is the bit width of configuration data in i-th field, and bit_width_{max} is available bit width in a multicasted data. S_{ijk} is non-zero value only if i-th field of the PE in the j-th row and k-th column has the same the configuration data which are considered to be multicasted. The first constraint ensures that required configuration data space does not exceed the available data space. The second constraint guarantees that the fixed configuration data are not overritten.

Given a multicasted configuration data, each S_{ijk} can be calculated. Therefore, S_{ijk}s are constant values. bit_width_i and bit_width_{max} are also constant. At first glance, the objective function is not linear. However, the product of binary valuables can be replaced with an additional variable and two constraints. For example, considering the product of $isField_0$, $isRow_0$ and $isCol_0$, it can be replaced with a new variable y_{000} with two constraints: $y_{000} \le isField_0 + isRow_0 + isCol_0 - 2$ and $3 * y_{000} \ge isField_0 + isRow_0 + isCol_0$. Thus, the objective function can be formulated as a linear function.

The ILP for each unfixed configuration data d is solved repeatedly. Unlike the *Espresso*-based algorithm, the ILP-based algorithm consists of only a loop because the combination of the fields and the bit-maps are decided at the same time.

An ILP is solved by the function "find_bitmap_by_ILP", then, $isRow$s, $isCol$s and $isField$s are respectively returned as B_r, B_c and f.

5. Evaluation

5.1 Evaluation Setup

First, we develop tools which can generate data for multicasting with the three methods: 1) the previous method, 2) fine grain multicasting with *Espresso* (FGM-E) and 3) fine grain multicasting with ILPs (FGM-I). The possibility of reducing the configuration data for each method is then evaluated. Evaluation environment is shown in Table 1. Both FGM-E and FGM-I have a high degree of data-level parallelism associated with the configuration fields. Therefore, we can easily parallelize them with multiprocessing package included in the Python standard library. We limit the size of multiprocess to 12 because it is enough size for these algorithms. Memory utilization of the algorithms is at most 600MB (FGM-I).

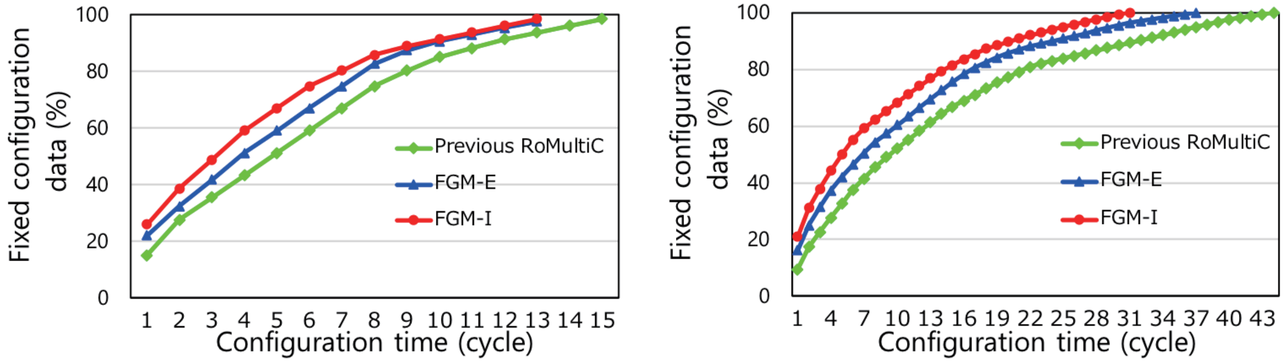
The drawback of *Espresso* depends on a mapping size of an application. For instance, when many PEs are utilized, *Espresso* is likely to fail the multicasting. In order to evaluate the influence, a lot of data-flow-graphs (DFGs) with various node size are generated randomly. Then, the multicasted data for each method are calculated. A range of PE utilization of the randomly generated DFGs is from 35% to 75%.

5.2 Reduction of the Configuration Data

Figure 9 describes how much the configuration data are multicasted with the three methods respectively. Two DFGs different in sizes are selected as examples. In case of a small DFG which utilizes 2 PE columns (Fig. 9 (a)), FGM-I finds the same or more multicasted data than FGM-E. Although both of them consequently take the same configuration time (13 cycles), they complete the configuration 2 cycles faster

Table 1 Evaluation environment

Implementation	
Programing	Python 3.4.5
Parallelization	multiprocessing
LP modeler	PuLP 1.6.5
ILP solvver	CBC 2.9.0
Espresso	PyEDA 0.28.0
Execution	
CPU	Intel Xeon CPU E5-2667 2.90GHz
Memory	128GB
Muximum process size	12



(a) 2 PE columns are used

(b) 6 PE columns are used

Fig. 9 Examples of fixed configuration data size during reconfiguration for each methods

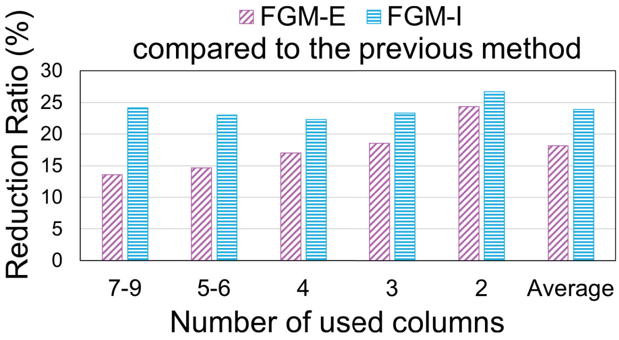


Fig. 10 Reduction ratios for each algorithm

Table 2 Application features

Application	Description	No. of used columns
gray	24 bit (RGB) gray scale	2
sepia	8 bit sepia filter	3
af	24 bit (RGB) alpha blender	4
sf	24 bit (RGB) sepia filter	3

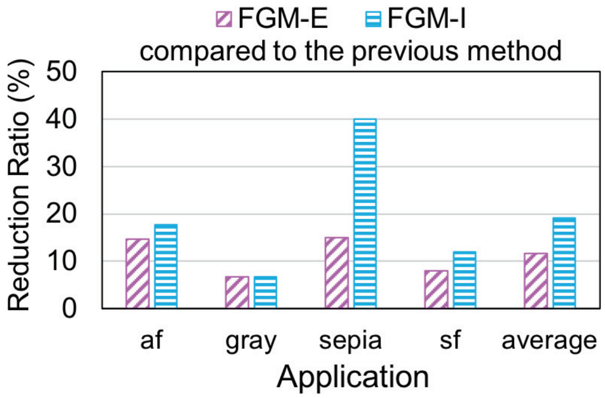


Fig. 11 Reduction effect on real applications

than the previous method. For another example, if 6 PE columns are used (Fig. 9 (b)), FGM-I also finds more multicasted data and finishes the configuration in 6 fewer cycles than FGM-E.

The average reduction ratio between the previous method [1] and both FGM-E and FGM-I is shown in Fig. 10. Compared to the previous method, the FGM-I achieves 23.8% reduction of configuration data in average. Originally, the previous method can reduce the configuration data by 60% in comparison with the single-cast method. As expected, the larger DFG is configured, the smaller reduction ratio of FGM-E can be observed. On the other hand, FGM-I achieves similar reduction ratio for all configurations.

Furthermore, three methods are applied to four image processing applications summarized in Table 2. Each of them does not require the whole PE array as described in Table 2. Therefore, the DFG of each application is duplicated on the PE array. For example, the DFG of *gray* can be duplicated 6 times.

The results are shown in Fig. 11. In case of *gray*, the reduction effect of FGM-E is not different from that of FGM-I, since it needs a small number of columns. However, compared to the case of random DFGs in Fig. 10, the reduction ratio is not so large.

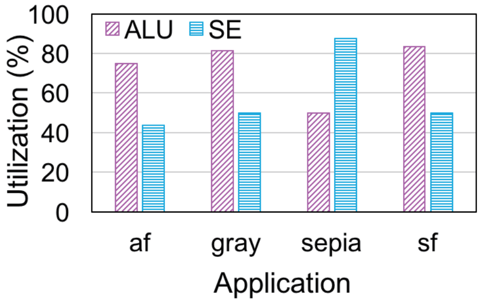


Fig. 12 Resource utilization for each application

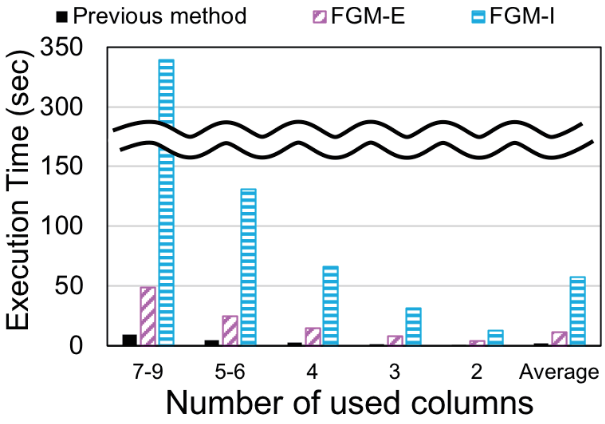


Fig. 13 Execution time for each method

The small reduction is associated with an utilization of SEs in the PEs. When a few SE is used, ALU configurations account for almost all of the configurations. Therefore, it is enough to multicast to the ALUs. The resource utilization for each application is shown in Fig. 12. Compared to other applications, *sepia* utilizes more than twice the SEs. In case of *sepia*, FGM-I achieves about 40% reduction due to high utilization of the SEs.

5.3 Time to Schedule the Configuration Data

Compilation time of the CGRAs is also important. In general, application mapping takes most of the compilation time, and it is completed within a few seconds to a few hours even by state-the-art mapping techniques [18]–[20]. For the multicasting, an additional compilation time in order to make the configuration data is required. Therefore, we evaluate the execution time for each algorithm.

Figure 13 depicts the execution time to obtain the multicasting data for each algorithm. Although FGM-I indicates better reduction ratio than FGM-I and the previous method regardless of the DFG size, it takes a long time to generate the multicasted data. When the large DFG is configured, FGM-E also takes a long time

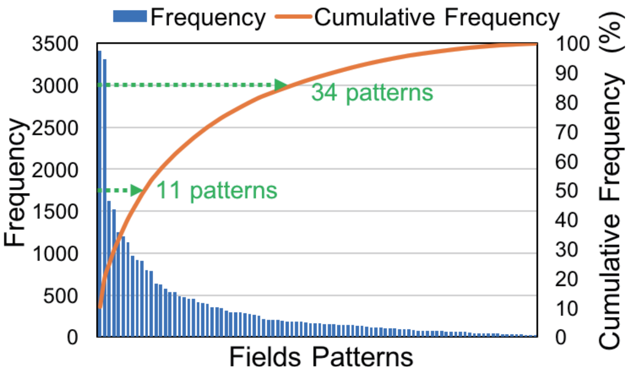


Fig. 14 Pareto chart of frequency in the use for each fields pattern

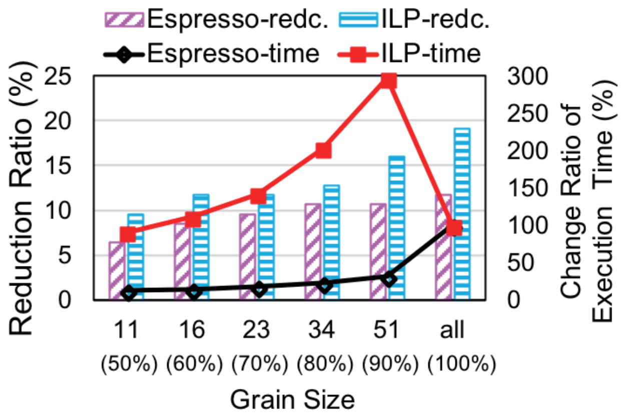


Fig. 15 Results for each grain size

Table 3 Top 11 frequently used field patterns

Rank	Pattern	Bit width	Frequency (%)
1	*NORTH*	3	10.2
2	*OP,SEL_A,SEL_B,SOUTH*	12	9.92
3	*OP,SEL_A,SEL_B*	10	4.86
4	*OP*	4	4.57
5	*OP,SEL_A,SEL_B,WEST*	12	3.75
6	*NORTH,WEST*	5	3.60
7	*EAST*	3	3.39
8	*OP,SEL_B*	7	2.90
9	*WEST*	2	2.74
10	*SOUTH*	2	2.73
11	*NORTH,SOUTH*	5	2.40

Table 4 Area overhead for each implementation

	area (mm^2)	overhead (%)
previous method	0.944	—
FGM	1.04	9.76
double buffer	1.73	73.1

Table 5 Power consumption during the configuration for each implementation

	Dynamic power (μW)	Static power (μW)
previous method	514.5	6.125
FGM	329.3	6.247
double buffer	546.2	11.76

5.4 Grain Size Exploration

Based on the results in Fig. 10, we analyzed how often each field pattern is used for multicasting. The results are shown in Fig. 14 and Table 3. They clearly show the frequencies of them are strongly biased. For example, the 11 most frequently used patterns account for about 50% usage. In order to cover 80%, it is enough to use 34 patterns.

For grain size exploration, we evaluated the reduction effect for each algorithm while changing a grain size, that is, the number of available field patterns. We evaluated size of 11, 16, 23, 34, and 51 corresponding to 50%, 60%, 70%, 80%, and 90% coverage of the frequencies, respectively. If the ILP is used, the limited field patterns make constraints of the ILP more complicated. As a result, it takes about 3.5 times longer time to solve an ILP problem. Thus, we modified the algorithm using ILP in order to shorten the execution time. Like the algorithm with *Espresso*, the modified ILP finds only the optimal bitmaps, and the optimal filed pattern is found by using the ILP iteratively. Here, the number of iteration is related to the grain size.

Figure 15 shows both the reduction effect and the execution time for each algorithm. In spite of the modification of the algorithm using ILP, its execution time is still long. In the worst case, the ILP for the grain size of 51 (90% coverage) takes 3 times longer execution time than that of the full support case (100% coverage). Besides, the reduction ratio is significantly decreased by increasing the grain size. Thus, in terms of both the reduction effect and the execution time, it is not effective to increase the grain size.

On the other hand, the execution time of the algorithm with *Espresso* is reduced greatly thanks to the smaller search space. In addition, the reduction ratio is close to the fine grain case when the grain size is higher than 34 (80% coverage). If only 34 patterns are used, the overhead for indicating the multicasted fields decreases to about one-third (34 / 94).

5.5 Overheads

The CC-SOTB architecture with the configuration controller which supports the fine grain multicasting is implemented using SOTB 65 nm process technology with Synopsys Design Compiler to analyze overheads of the proposed method. For comparison, another method with a double buffer is also implemented. Using the double buffer like a context switching is another way to address the long reconfiguration time. Instead of reducing the transferred data, it aims to hide the latency of reconfiguration.

Area overhead for each implementation is shown in Table 4. Twice the size of configuration registers causes a large area overhead while the overhead of the proposed FGM is not too large.

In addition, Table 5 describes a power consumption for each implementation. The power consumption is evaluated with Synopsys PrimeTime and Cadence NC-Verilog. The

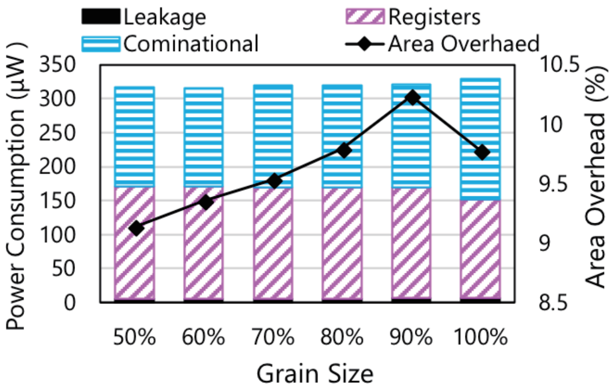


Fig. 16 Power consumption and area overhead for each grain size

operating frequency and the supply voltage are set to 30 MHz and 0.55 V, respectively. Both the previous method and the method with the double buffer consume almost the same dynamic power. However, the double buffer consumes about two times of the static power. On the other hand, the proposed method achieves 35.6% lower dynamic power consumption than the previous method. This is because writing the configuration data finely minimizes unnecessary switching in the configuration registers. Considering the reconfiguration time shortened by 23.8% in average, 50.1% reduction of energy consumption can be achieved.

In the previous subsection, we analyze how much changing the grain size affects the reduction ratio and the execution time. Next, we implement configuration decoder for each grain size. Then, hardware overhead for each grain size is evaluated. In the full support case (100%), as explained in Sect. 3.1, the flag bitmap is used in order to specify which fields the data contain. However, in the reduced cases such as "50%", the bitmap manner is too excessive. Hence, an LUT-based decoders are implemented for the reduced cases. For example, in the "50%" grain size, 11 field patterns are used so that it requires only 4 bits to distinguish them. Please note that the 4 bits are also contained in the address space like the flag bitmap.

The average power consumption of the four applications for each grain size is shown in Fig. 16. The power is calculated at the same condition as Table 5. The results reveal the reduced grain size does not contribute to the power reduction. Because of the difference of decoder types, the full support needs a little more power consumption of combinational circuits. In contrast, it saves the power consumption of the configuration registers. The area overheads are also shown in Fig. 16. The result suggests that the LUT-based decoders from 50% to 70% are effective. In other cases, the flag bitmap method is better. However, the difference in the area overheads is not so large.

6. Conclusion and Future Work

In this work, we have introduced a new configuration multicasting scheme for CGRAs. By optimizing the grain of multicasting, the proposed method can reduce both the configuration data and the dynamic power consumption. In order to generate multicasted data, two algorithms based on *Espresso* and ILP respectively are considered. As the experimental results, they provide a possibility of a trade-off between the reduction ratio and the execution time. When the proposed method is applied to real applications, about 40% reduction of the configuration data can be achieved in the best case. Furthermore, the energy consumption can be reduced by nearly half.

In the evaluation of this study, relatively small DFG applications are employed. Therefore, an evaluation with more complicated and larger DFGs will be performed in future work. In addition, we should assess a reduction of an application runtime as well as the configuration time reduction.

Acknowledgments

This work is partially supported by JSPS KAKENHI S Grant Number 25220002 and JSPS KAKENHI B Grant Number 18H03215. This work is supported by VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Synopsys, Inc and Cadence Design Systems, Inc.

References

[1] S. Tsutsumi, V. Tunbunheng, Y. Hasegawa, A. Parimala, T. Nakamura, T. Nishimura, and H. Amano, "Overwrite configuration technique in multicast configuration scheme for dynamically reconfigurable processor arrays," 2007 International Conference on Field-Programmable Technology, ICFPT, pp.273–276, IEEE, 2007.
[2] S.M.A.H. Jafri, A. Hemani, K. Paul, J. Plosila, and H. Tenhunen, "Compression based efficient and agile configuration mechanism for coarse grained reconfigurable architectures," 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), pp.290–293, IEEE, 2011.
[3] R.K. Brayton, G.D. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli, Logic minimization algorithms for VLSI synthesis, Kluwer Academic Publishers, Boston, 1984.
[4] N. Ozaki, Y. Yasuda, M. Izawa, Y. Saito, D. Ikebuchi, H. Amano, H. Nakamura, K. Usami, M. Namiki, and M. Kondo, "Cool Mega-Arrays: Ultralow-Power Reconfigurable Accelerator Chips," IEEE Micro, vol.31, no.6, pp.6–18, Nov. 2011.
[5] Y. Matsushita, H. Okuhara, K. Masuyama, Y. Fujita, R. Kawano, and H. Amano, "Body bias grain size exploration for a coarse grained reconfigurable accelerator," 2016 26th International Conference on Field Programmable Logic and Applications (FPL), pp.1–4, Aug. 2016.
[6] S. Park and K. Choi, "An approach to code compression for CGRA," 2011 3rd Asia Symposium on Quality Electronic Design (ASQED), pp.240–245, IEEE, 2011.
[7] B. Liu, W.-Y. Zhu, Y. Liu, and P. Cao, "A configuration compression approach for coarse-grain reconfigurable architecture for radar signal processing," 2014 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), pp.448–453, IEEE, 2014.
[8] T. Kojima and H. Amano, "A configuration data multicasting method for coarse-grained reconfigurable architectures," 2018 28th International Conference on Field Programmable Logic and Applications (FPL), pp.239–2393, IEEE, 2018.
[9] B. Egger, H. Lee, D. Kang, M.S. Moghaddam, Y. Cho, Y. Lee, S.

Kim, S. Ha, and K. Choi, "A space-and energy-efficient code compression/decompression technique for coarse-grained reconfigurable architectures," Proceedings of the 2017 International Symposium on Code Generation and Optimization, pp.197–209, IEEE Press, 2017.

[10] M.-K. Chung, J.-K. Kim, Y.-G. Cho, and S. Ryu, "Adaptive compression for instruction code of coarse grained reconfigurable architectures," 2013 International Conference on Field-Programmable Technology (FPT), pp.394–397, IEEE, 2013.

[11] Y. Kim and R.N. Mahapatra, "Dynamic context compression for low-power coarse-grained reconfigurable architecture," IEEE transactions on very large scale integration (VLSI) systems, vol.18, no.1, pp.15–28, 2010.

[12] X. Qin, C. Muthry, and P. Mishra, "Decoding-aware compression of FPGA bitstreams," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol.19, no.3, pp.411–419, 2011.

[13] S.-W. Seong and P. Mishra, "Bitmask-based code compression for embedded systems," IEEE Transactions on computer-aided design of integrated circuits and systems, vol.27, no.4, pp.673–685, 2008.

[14] Y. Gao, H. Ye, J. Wang, and J. Lai, "FPGA bitstream compression and decompression based on LZ77 algorithm and BMC technique," 2015 IEEE 11th International Conference on ASIC (ASICON), pp.1–4, IEEE, 2015.

[15] R. Iša and J. Matoušek, "A novel architecture for LZSS compression of configuration bitstreams within FPGA," 2017 IEEE 20th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS), pp.171–176, IEEE, 2017.

[16] J.S. Kumar, G.S. Kumar, R. Kannan, A. Varatharaj, J. Lawrence, S. Krishnamoorthi, N. Sandhiya, L. Subathra, R. Vinothini, and S. Sonu, "Bit stream compression used for FPGA using Golomb coding," 2013 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), pp.1–4, IEEE, 2013.

[17] S. Hauck, Z. Li, and E. Schwabe, "Configuration compression for the Xilinx XC6200 FPGA," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol.18, no.8, pp.1107–1113, 1999.

[18] D. Liu, S. Yin, G. Luo, J. Shang, L. Liu, S. Wei, Y. Feng, and S. Zhou, "Data-Flow Graph Mapping Optimization for CGRA with Deep Reinforcement Learning," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2018.

[19] S. Yin, X. Yao, D. Liu, L. Liu, and S. Wei, "Memory-aware loop mapping on coarse-grained reconfigurable architectures," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol.24, no.5, pp.1895–1908, 2016.

[20] S. Dave, M. Balasubramanian, and A. Shrivastava, "RAMP: resource-aware mapping for CGRAs," Proceedings of the 55th Annual Design Automation Conference, p.127, ACM, 2018.

Takuya Kojima received BS degree from Keio University, Yokohama, Japan, in 2017. He is a master student in Keio university in the presence.

Hideharu Amano received Ph.D degree from the Department of Electronic Engineering, Keio University, Japan in 1986. He is currently a professor in the Department of Information and Computer Science, Keio University. His research interests include the area of parallel architectures and reconfigurable systems.