SAT Based Formulation of Automatic Generation of Parallel Computing from Specification

Gao Ruitao[†1] Amir Masoud Gharehbaghi[†1]
Tomohiro Maruoka[†1] Masahiro Fujita[†1]

In recent years, methods using deep learning have been widely used in various fields. And It is known that a large portion of computation time in deep neural network is taken by matrix multiplication. There is close connection between neural network and matrix multiplication. In this paper, parallel computing solution for matrix-vector multiplication on certain ring-connected cores is automatically generated. The basic method is to formulate matrix-vector multiplication on ring-connected architecture as a SAT problem and use SAT solver to get the mapping solution. According to the experiment results, parallel computing solution of 16x16 matrix can be generated in short time. Moreover, solutions for sparse matrix multiplication can be generated.

1. Introduction

In recent years, methods using deep learning have been widely used in various fields. For example, in the field of image recognition, the methods using deep learning outperform other methods. numerous deep neural networks such as VGG net, GoogLeNet, and ResNet have shown excellent performance. [1] [2]

It is known that about 90% of the computation time in the deep neural network is taken in the convolution layer. Thus, accelerating and optimizing the matrix multipliers used in convolution can be benefitial.

Parallel computing is an effective approach to reduce computing time. And in recent years, more and more computations have been performed with multicore CPU, GPU, and FPGA. There is diversity in the communication structure among nodes (cores or chips), such as ring [3][4], mesh [5], and others.

In this paper, we have proposed a method to automatically map matrix-vector multiplication on certain ring-connected architecture to realize parallel computing. A ring-connected architecture with special structure is defined for matrix-vector multiplication, and formulation of matrix-vector multiplication on ring-connected architecture are introduced. For sparse matrices which are common in real neural network, we modify the formulation to fit matrix-vector multiplication with sparse matrix. Then, we transform formulation into a SAT problem in form of a BLIF (Berkeley Logic Interchange Format) file [6] and input it into ABC [7] to get the mapping solution. ABC is a public-domain system for logic synthesis and formal verification of binary logic circuits appearing in synchronous hardware designs. It can transform BLIF file into CNF (Conjunctive Normal form) and internally solve it with a SAT solver. From the SAT solver, we can get the arrangement of variables to generate mapping solution. A program is written to realize automatic generation of mapping solution. Several experiments for dense matrix and sparse matrix are conducted respectively. Times to generate mapping solution under different circumstances with different parameters and examples of mapping solutions are presented.

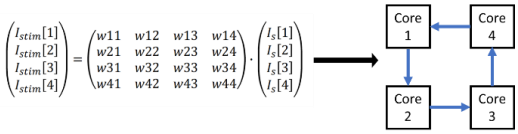

figure 1 matrix-vector multiplication on ring-connected architecture

The paper is organized as follows. In section 2, related works of matrix-vector multiplication and features of this research are introduced. In section 3, approach of the research is introduced. Section3.1 is about features of ring-connected architecture and structure inside each core. Section 3.2 explains the formulation. Section 3.3 introduces method of transformation. Section 4 presents the experimental results. Section 5 concludes the paper.

2. Related works

Several studies have been conducted about matrix-vector multiplication. Nathan Bell and Michael Garland discussed data structures and algorithms for sparse matrix-vector multiplication implemented on the CUDA platform and research for method to implement matrix-vector multiplication on throughput-oriented Processors [8][9]. Xia, Lixue, et al explored hardware realization of the analog matrix-vector multiplication with ultra-high energy efficiency on RRAM Crossbar Array. [10] Liu, Weifeng, and Brian Vinter proposed a sparse matrix-vector multiplication algorithm utilizing both types of cores in a CPU–GPU heterogeneous processor. [11] All these researches focus on matrix-vector multiplication on existing architectures, but in this paper, we proposed a ring-connected architecture specially defined for matrix-vector multiplication and try to generate matrix-vector multiplication on it.

There has also been a research of matrix-vector multiplication on specially designed architectures. [12] In the research, matrix-vector multiplication on a new reconfigurable architecture based on ILP formulation has been realized. Mapping for 4×4 matrix is first generated, then mapping for arbitrary size matrices similar to the case 4×4 can be generated with more constraints. Matrices here are dense matrices where most of the elements are nonzero. However, as we know, in deep neural network, most of matrices in matrix-vector multiplication are sparse matrices in which a large proportion of the elements are zero

In this paper, we can realize automatic generation of matrix-

†1 University of Tokyo

vector multiplication with arbitrary matrix size from 2×2 to 16×16 without generation of smaller matrix mapping result or additional constraints. Also, we take not only dense matrix but also sparse matrix into consideration. Matrix-vector multiplication with sparse matrix can be mapped on ring-connected architecture in fewer time cycles.

3. Approach

In this paper, we first define a special ring-connected architecture and then come up with formulation of mapping matrix-vector multiplication with dense matrix and with sparse matrix on the ring-connected architecture, transform it into a SAT problem. Finally, SAT solver in ABC is used to determine satisfiability of mapping. If the result is SAT, we get the mapping solution.

3.1 Ring-Connected Architecture

In this research, the architecture to be mapped on is a ring-connected architecture with certain number of cores. In the schematic diagram in figure 1, a ring-connected with 4 cores is presented. In this architecture, in every time cycle, cores work in parallel to realized distributed computing [13]. Data transfer here has directionality. Data can be transferred from one core to the next one but the transmission to the last core is prohibited. For example, we can transfer data from core1 to core2 and core4 to core 1, but we can't get data from core2 to core1 or from core1 to core4. Also, at the end of every time cycle for each core, no more than one data can be transferred to the next core.

Figure 2 explains the structure inside each core. In each core, there are several registers and an ALU (Arithmetic Logical Unit). One multiplication and one addition can be performed in ALU in every time cycle. And the sum of products from ALU is stored into the register.

Is represents element in vector of multiplication and Istim represents element in result vector. $w[y][x]$ from the input matrix and $Is[x]$ from the vector are mapped on the cores before first time cycle. Before the last time cycle, $Istim[y]$ represents the sum of product up to now for each $Istim[y]$, and in the last time cycle, that is the result of the multiplication. For example, for the matrix-vector multiplication [11] in figure 1. The $Istim[1]$ should be:

$$Istim[1] = w11Is[1] + w12Is[2] + w13Is[3] + w14Is[4]$$

Assume that at time cycle 2, w11 is multiplied by $Is[1]$ and then added by $w13Is[3]$, the result $\mathrm{w11}Is[1] + w13Is[3]$ is stored in a register, and that $\mathrm{w11}Is[1] + w13Is[3]$ is our $Is[1]$ at time cycle 2. In every time cycle, there's no more than one data of $Is[x]$or $Istim[y]$ comes from the last core. The received data is stored into one of the registers. After that, certain $Is[x1]$ and $w[y][x1]$ from registers are input into the multiplier of ALU. and the product $w[y][x1]$ is added by the third input, $Istim[y]$ of last time cycle. The output of ALU, the new $Istim[y]$, is stored into the register. At the end of each time cycle, there can be no more than one data of $Is[x]$ or $Istim[y]$ from register transferred to the next core. At the last time cycle, the vector which is consist of $Istim[y]$s is the result of matrix-vector multiplication.

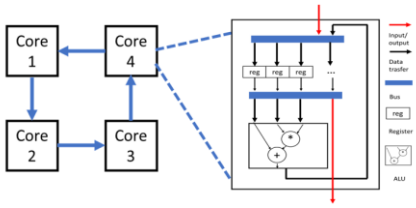

figure 2 ring-connected architecture and structure inside each core

3.2 Formulation

To correctly perform the matrix-vector multiplication on the aforementioned ring-connected architecture, several kinds of variables representing the mapping status of data and several kinds of constraints describing manner of data and cores are necessary.

Formulation of matrix-vector multiplication with dense matrix will be introduced first.

3.2.1 For Dense Matrix

There are 6 kinds of binary variables. $\mathrm{w}[y][x](t,c)$ decides whether $\mathrm{w}[y][x]$ is used on core c at time cycle t; $Is[x](t,c)$ decides whether $Is[x]$ is mapped on core c at time cycle t; $Istim[y](t,c)$ decides whether $Istim[y]$ is mapped on core c at time cycle t; $Is\_next[x](t,c)$ decides whether $Is[x]$ moves from core c to core c+1 at time cycle t; $Istim\_next[y](t,c)$ decides whether $Istim[y]$ moves from core c to core c+1 at time cycle t and $\mathrm{w}[y][x]Is[x](t,c)$ decides whether multiplication $\mathrm{w}[y][x] \times Is[x]$ is executed on core c at time cycle t. For all these binary variables, when the value is 1 it means that the behavior defined by the variables happens and when the value is 0 it means that the behavior defined by the variables doesn't happen. $Is\_next[x](t,c)$ and $Istim\_next[y](t,c)$ represent whether there are transmissions, and the other variables represent behaviors of registers.

Here's an example of definition of variables in figure 3. This is the arrangement of variables to map the following matrix-vector multiplication on 2 interconnected cores.

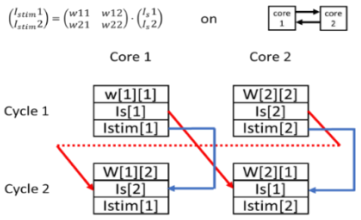

figure 3 Example of Definition of Variables

Here, $w[1][1]$ is used on core 1 at time cycle 1, so $w[1][1](1,1)$ is 1; $Is[1]$ is mapped on core 1 at time cycle 1, so $Is[1](1,1)$ is 1; multiplication $w[1][1] \times Is[1]$ is executed on core 1 at time cycle 1, and $w[1][1]Is[1](1,1)$; $Istim[2]$ is mapped on core 1 at cycle 2, so $Istim[2](2,1)$ is 1; $Is[1]$ moves from core 1 to core 2 after time cycle 1, so $Is\_next[1](1,1)$ is 1 and $Istim[2]$ doesn't move after time cycle 1, so $Istim\_\mathrm{next}[2](1,2)$ is 0.

For these binary variables, to correctly perform matrix-vector multiplication on ring-connected architecture, there are several constraints. According to their functions, constraints can be divided into 4 groups.

Mapping constraints are in the first group and describe mapping behaviors of $w[y][x]$, Is[x] and $Istim[y]$. w mapping

constraints set that $w[y][x]$ must be used once on somewhere through all cycles and is (1).

$$\forall x, y \sum_t \sum_c w[y][x](t,c) = 1 \quad (1)$$

Is mapping constraints ensure $Is[x]$ to exist once on somewhere every cycle and is shown in (2).

$$\forall x, t \sum_c Is[x](t,c) = 1 \quad (2)$$

With Istim mapping constraints $Istim[y]$ must be mapped once on somewhere every cycle and the constraints are (3).

$$\forall y, t \sum_c Istim[y](t,c) = 1 \quad (3)$$

wIs mapping constraints ensure multiplication $w[y][x] \times Is[s]$ must be executed once on somewhere through all cycles, and is (4).

$$\forall x, y \sum_t \sum_c w[y][x] Is[x](t,c) = 1 \quad (4)$$

The last kind of mapping constraints are IsIstim mapping constraints. It defines that on each core, $Is[x](1,c)$ is equal *to* $Istim[y](T,c)$ (T is the last time cycle). And that means x must be equal to y. The constraints are (5).

$$\forall x, y, c\ Is[x](1,c) - Istim[y](T,c) = 0 \quad (5)$$

Data transfer constraints describe data transfer of $Is[x]$ and $Istim[y]$ from core c to core c+1. when $Is[x](t,c)$ exists, Is transfer constraints must be satisfied. They are shown in (6)(7).

$$\forall x, t, c\ 0 \le -Is[x](t,c) + Is[x](t+1,c) + Is[x](t+1,c+1) \le 2 \quad (6)$$

$$\forall x, t, c\ 0 \le -Is[x](t,c) + Is[x](t-1,c) + Is[x](t-1,c-1) \le 2 \quad (7)$$

Figure 4 gives explanation of Is transfer constraints. Because in every time cycle, no more than one of $Is[x]$ and $Istim[y]$ can be transferred from core c-1 to core c and from core c to core c+1. If $Is[x](t,c) = 1$, which means that $Is[x]$ is mapped on core c at time cycle t, at time cycle t+1 $Is[x]$ must be mapped at core c or core c+1, $Is[x](t+1,c)$ or $Is[x](t+1,c+1)$ must be 1. Also, if $Is[x](t,c) = 1$, $Is[x]$ must be mapped on core c at time cycle t-1 or mapped on core c-1 at cycle t-1, $Is[x](t-1,c)$ or $Is[x](t-1,c-1)$ must be 1. In the above two situations, left part of (6) and (7) is 0. If $Is[x](t,c) = 0$, which means that $Is[x]$ is not mapped on core c at time cycle t If $Is[x](t+1,c+1)$ can be 1 because $Is[x](t,c+1)$ could be 1, and $Is[x](t-1,c-1)$ can be 1 because $Is[x](t,c-1)$ could be 1. When $Is[x](t+,c+1)$ and $Is[x](t-1,c-1)$ is one, left part of (6) and (7) is 1.

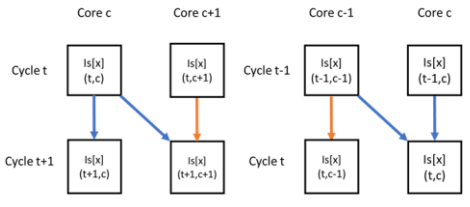

figure 4 Explanation of Is Transfer Constraints

Istim transfer constraints can be inferred from Is transfer constraints and the explanation is also similar. Istim transfer constraints are (8) and (9).

$$\forall y, t, c\ 0 \le -Istim[y](t,c) + Istim[y](t+1,c) + Istim[y](t+1,c+1) \le 2 \quad (8)$$

$$\forall y, t, c\ 0 \le -Istim[y](t,c) + Istim[y](t-1,c) + Istim[y](t-1,c-1) \le 2 \quad (9)$$

Is_next constraints and Istim_next constraints describe the relationships which must be met when and $Is\_next[x](t,c)$ and $Istim\_next[y](t,c)$ exist. Is_next constraints are (10) and Istim_next constraints are (11)

$$\forall x, t, c\ 0 \le -2 \times Is\_next[x](t,c) + Is[x](t,c) + Is[x](t+1,c+1) \le 1 \quad (10)$$

$$\forall y, t, c\ 0 \le -2 \times Istim\_next[y](t,c) + Istim[y](t,c) + Istim[y](t+1,c+1) \le 1 \quad (11)$$

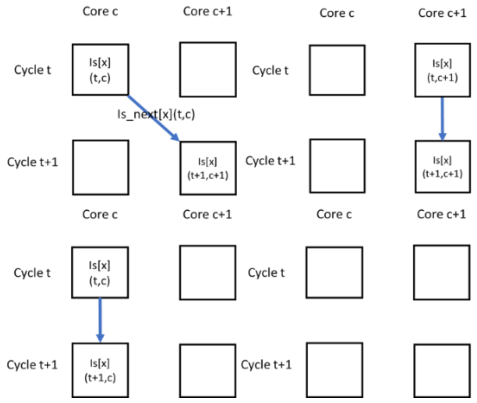

figure 5 Possible Situations in Is_next constraints

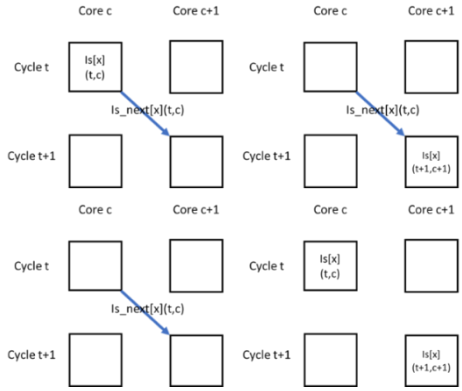

Figure 6 Impossible Situations in Is_next constraints

Figure 5 and figure 6 describe the relationship of $Is[x](t,c)$ and $Is\_next[x](t,c)$ in detail. Figure 5 shows the possible situations in Is_next constraints according to the definition of $Is\_next[x](t,c)$ and figure 6 shows the impossible situations. From figure 5 and figure 6, we can conclude that algebraic part of (10) can't be smaller than 0 or bigger than 1. Istim_next constraints have similar principle. Thus, algebraic part of (10) can't be smaller than 0 or bigger than 1.

The next part is about sum of products constraints. As we know, the final result of matrix-vector multiplication, $Istim[y]$ at the last time cycle, are all sum of products. To ensure operations in all cores at every time cycle have necessary data, we have sum of products constraints, which means $Is[x](t,c)$, $Istim[y](t,c)$ and $w[y][x](t,c)$ must be 1 when $w[y][x]Is[x](t,c)$ is 1. Sum of products constraints are (12)

$$\forall x, y, t, c\ 0 \le -3 \times w[y][x](t,c) + w[y][x](t,c) + Is[x](t,c) + Istim[y](t,c) \le 3 \quad (12)$$

The last part of constraints are resource constraints. Resource constraints come from the ring-connected architecture and restrict hardware resources can be used for mapping and data transfer.

ALU resource constraints specify that no more than one

multiplication $w[y][x] \times Is[x]$ can be executed on each core at every time cycle, and are (13)

$$\forall t, c\ 0 \le \sum_x \sum_y w[y][x] Is(t,c) \le 1 \quad (13)$$

Register resource constraints ensure that no more than $\lceil (X+Y) \div C \rceil$ $Is[x]$ and $Istim[y]$ can be mapped on each core at every cycle(C is number of cores). With register resource constraints, we can avoid wasting registers, only necessary number of registers are used. Register resource constraints are (14)

$$\forall t, c\ 0 \le \sum_x Is[x](t,c) + \sum_y Istim[y](t,c) \le \lceil (X + Y) \div C \rceil \quad (14)$$

From introduction of ring-connected architecture, we know that no more than one data can be transferred to the next core for each core every cycle. Thus, no more than one $Is[x](t,c)$ and $Istim[y](t,c)$ can be mapped on each core at every cycle. Edge resource constraints ensure this, and are shown in (15)

$$\forall t, c\ 0 \le \sum_x Is\_next[x](t,c) + \sum_y Istim\_next[y](t,c) \le 1 \quad (15)$$

For a correct mapping of multiplication on ring-connected architecture, all the constraints from (1) to (15) are necessary.

3.2.2 For Sparse Matrix

In real deep neural network, usually, a large proportion of elements are zero and results of multiplication with these elements are bound to be zero. Thus, we don't need to conduct these multiplications and add them to sum of products. Formulation for matrix-vector multiplication with sparse matrix can be obtained by modification of formulation for dense matrix.

$$\begin{pmatrix} I_{stim}[1] \\ I_{stim}[2] \\ I_{stim}[3] \\ I_{stim}[4] \end{pmatrix} = \begin{pmatrix} w11 & 0 & w13 & w14 \\ 0 & w22 & 0 & w24 \\ w31 & 0 & 0 & 0 \\ 0 & w42 & w43 & w44 \end{pmatrix} \cdot \begin{pmatrix} I_s[1] \\ I_s[2] \\ I_s[3] \\ I_s[4] \end{pmatrix}$$

figure 7 example of sparse matrix-vector multiplication

Compared with matrix-vector multiplication with dense matrix, in case of matrix-vector multiplication with sparse matrix, $Is[x]$s and $Istim[y]$s are the same, and the only difference is that some of w[y][x]s are bound to be 0. Thus, we just remove some of variable $w[y][x](t,c)$s and $w[y][x]Is[x](t,c)$s whose corresponding w[y][x]s are bound to be 0, and modify w mapping constraints, wIs mapping constraints, Sum of product constraints and ALU resource constraints where w[y][x]s exist. For w mapping constraints, wIs mapping constraints and Sum of product constraints, we directly remove the all the constraints where aforementioned $w[y][x](t,c)$ or $w[y][x]Is[x](t,c)$ exists. For ALU resource constraints, the aforementioned $w[y][x]Is[x](t,c)$ s are removed from the constraints. The modified formulation is a correct description of behavior of matrix-vector multiplication on the ring-connected architecture.

3.3 Transformation into BLIF file

With given number of cores in the ring-connected architecture c, number of time cycle t and size of matrix x, y, mapping matrix-vector multiplication on the ring-connected architecture can be regarded as a SAT problem. SAT problem can be solved by a SAT solver, and the satisfiability of can be determined. In order to use the SAT solver in ABC, we translate the formulation into BLIF file and use the BLIF file as the input of SAT solver.

BLIF file describes behavior of circuit. For (2), Is mapping constraints. From the view of Boolean function, is means that only and must one of the variables in left part is 1. Here is an example of translation when x = y = t = c = 4 in Figure 8.

```
.names is[1](1,1) is[1](1,2) is[1](1,3) is[1](1,4) is_mapping_constraint1
0001 1
0010 1
0100 1
1000 1
```

figure 8 example of translation of Is mapping constraints

The exception is IsIstim constraint, (5). To translate this kind of constraint is to write XNOR gate. You can see in figure 9, only when Is[1](1,1) and Is[1](4,1) are equal, output is 1, which means the constraint is satisfied.

```
.names is[1](1,1) istim[1](4,1) is&istim_mapping_constraint1
11 1
00 1
```

figure 9 example of translation of IsIstim mapping constraints

For Is transfer constraints and Istim transfer constraints, we have introduced in detail in figure 4. We record all the possible situations to be 1 and ignore others. An example in presented in figure 10. ("-" means don't care)

```
.names is[1](1,1) is[1](2,1) is[1](2,2) is_transfer_constraint1
-1- 1
--1 1
0-- 1
```

figure 10 example of translation of Is transfer constraints

For Is_next constraints and Is_next constraints, we have similar operations, just write down all arrangements of variables which satisfy the constraints. An example is shown in figure 11.

```
.names is_next[1](1,1) is[1](1,1) is[1](2,2) is_next_constraint1
111 1
010 1
001 1
000 1
```

figure 11 example of translation of Is_next constraints

Sum of product constraints can be translated according to its meaning. This kind of constraints ensure operations in all cores at every time cycle have necessary data. Thus, to satisfy the constraints, when w[y][x](t,c)=1, other variables should be 1; if w[y][x](t,c)=0, we don't care about other variables. Figure 12 is an example.

```
.names w[1][1]is[1](1,1) w[1][1](1,1) is[1](1,1) istim[1](1,1) sum_of_product_constraint1
0--- 1
1111 1
```

figure 12 example of sum of product constraints

Resource constraints restrict the sum of value of variables in left part. Thus, to satisfy the constraints, no more than the value of right part can be 1, and Figure 13 is an example of register resource constraints.(x=y=t=c=2)

```
.names is[1](1,1) is[2](1,1) istim[1](1,1) istim[2](1,1) register_resource_constraint1
0000 1
0001 1
0010 1
0011 1
0100 1
0101 1
0110 1
1000 1
1001 1
1010 1
1100 1
```

figure 13 example of register resource constraints

SAT means that the mapping problem is feasible, vice versa. When the result is SAT, the SAT solver can output the certain combination of values of BLIF file's input parameters. With these values of BLIF file's input parameters, mapping solution can be generated.

4. Experiments

Figure 14 explains experiment process in the research. First, corresponding formulation of input parameters (size of matrix x, y, time cycle t and number of cores c) is generated. For sparse matrix, positions of 0 elements are also collected. Then, the

formulation is transformed into a BLIF file. After that, the BLIF file is put into SAT solver in ABC to determine satisfiability. Finally, mapping solution is generated when the result is SAT.

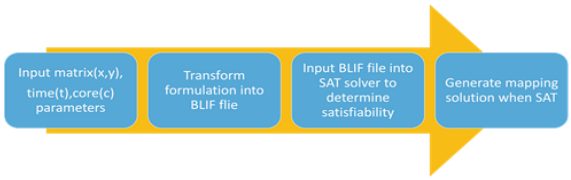

figure 14 Experimental flow chart

We write a program to perform these steps automatically. With proper input parameters, satisfiability of mapping corresponding matrix-vector multiplication on ring-connected architecture can be determined, and mapping solution can be generated when the result is SAT.

4.1 For Dense Matrix

The purpose is to generate mapping solution. Thus, the first thing is to determine the prerequisites that ensure SAT. For a matrix-vector multiplication where the size of matrix is x,y, $x \times y$ multiplication of $w[y][x]Is[x]$ should be executed and there are only $c \times t$ times of operation available because no more than 1 operation can be executed for each core in every time cycle according to the ring-connected architecture. when $t < x \times y/c$ some of the multiplication are sure to not be mapped, Thus, it is bound to be UNSAT and we can't get mapping solutions. When $t \geq x \times y/c$, there are enough ALU for multiplication and operation, but other constraints should also be satisfied to ensure SAT. The subsequent experiments are conducted in case of $t \geq x \times y/c$.

4.1.1 When Size of Matrix is Divisible by Number of Cores

When size of matrix(x,y) is divisible by number of cores c, theoretically, x/c time cycles is needed and ALUs will be fully utilized in every time cycle, the mapping result is expected to be regular. Experiment results when x,y is divisible by c is presented in table 1. Matrices with size from 2×2 to 17×17 can be mapped on the ring-connected architecture within relatively short time, solution for 16×16 matrix can be generated in 280s.

matrix size	number of cores	time cycles	execution time(s)
2×2	2	2	0.00
3×3	3	3	0.01
4×4	4	4	0.01
4×4	2	8	0.02
6×6	3	12	0.12
6×6	6	6	0.05
7×7	7	7	0.09
8×8	4	16	1.92
8×8	8	8	0.28
10×10	10	10	3.49
12×12	12	12	3.02
13×13	13	13	5.94
16×16	16	16	279.13
17×17	17	17	911.03

table 1 results when x,y is divisible by c

With the automatically generated mapping solution, parallel computing of matrix-vector multiplication can be realized on the ring-connected architecture. Theoretically, conduct 16×16 matrix-vector multiplication on 16 ring-connected cores can accelerate computing speed by 15 times. Figure 15 is a mapping solution example when $x = y = t = c = 4$. All ALUs are fully utilized and the mapping solution is regular.

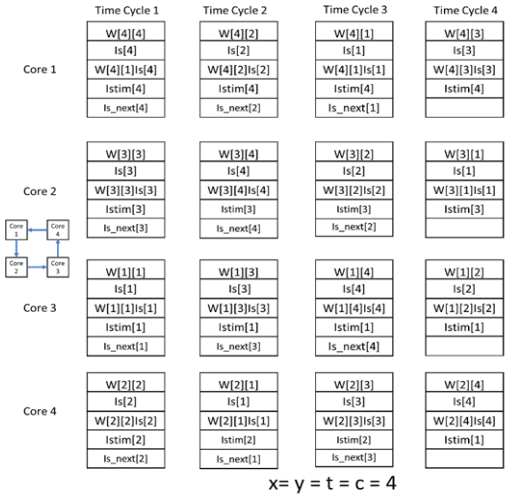

figure 15 mapping solution example when x, y is divisible by c

4.1.2 When Size of Matrix is Indivisible by Number of Cores

When size of matrix(x,y) is indivisible by number of cores c, theoretically, more than x/c time cycles is needed and some of ALUs will be idle in certain time cycle, the mapping result is expected to be irregular. Experiment results when x,y is divisible by c is presented in table 2.

matrix size	number of cores	time cycles	execution time(s)
3×3	2	5	0.01
4×4	3	6	0.02
5×5	3	9	0.04
5×5	4	7	0.10
6×6	4	9	0.10
7×7	4	13	0.36
7×7	5	10	11.32
8×8	5	13	11.48
8×8	6	11	826.53
9×9	5	17	5.88
9×9	6	14	207.72
10×10	8	13	421437.38

table 2 results when x,y is indivisible by c

Compared with data in table 1, for matrix with the same size, much more time is needed to generate mapping solution when x,y is indivisible by c. The mapping solutions tend to be irregular and more difficult to generate. Figure 16 is a mapping solution example when x=y=3, t=5, c=2. ALU in core is idle in time cycle 1, ALU utilization is lower when x,y is indivisible by c.

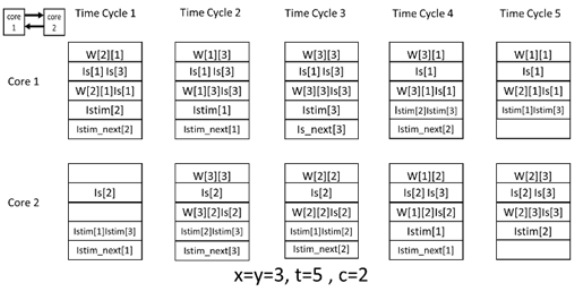

figure 16 mapping solution example when x, y is indivisible by c

4.2 For Sparse Matrix

Theoretically, with the same x, y, c, compared to dense matrix, fewer multiplications and additions are needed to be conducted

for sparse matrix, and fewer time cycles are needed to complete the matrix-vector multiplication. However, situations where many non-zero elements are in the same column should be taken into consideration. According to Is mapping constraints (2), certain $Is[x_0]$ exist only once every cycle, but all $w[y][x_0]$s must multiply with $Is[x_0]$ once. If there's m non-zero $w[y][x_0]$s in the same column, the mapping is infeasible for $t < m$. Assume t_1 is theoretical minimal number of time cycles, then$t_1 = max\{m, \lceil x \times y - n/c \rceil\}$. Here, m is the maximum of number of non-zero $w[y][x_0]$s in the same column, and n is number of zero elements.

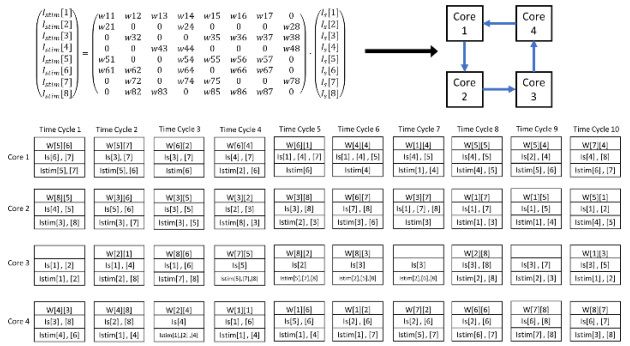

figure 17 mapping solution for sparse matrix

Figure 17 presents an example of mapping solution for sparse matrix. For mapping the matrix-vector multiplication in figure 12 on 4 ring-connected cores $t_1 = max\{6, \lceil 8 \times 8 - 27/4 \rceil\} = 10$. If we use the formulation for dense matrix, we can also get a feasible mapping solution with 16 time cycles. But with formulation modified for sparse matrix, the same matrix-vector multiplication can be completed in 10 time cycles. Modified formulation for sparse matrix helps improve ALU utilization in mapping solution.

To explore the effect of number of zero elements, the following experiment is conducted. A Matrix-vector multiplication with 8×8 sparse matrix is to be mapped on 4 ring-connected cores. Number of zero elements in the matrix is changed during experiment. Table 3 presents the result. To reduce the error, operation time here is average value of 10 times experiments. It is obvious that with more zero elements, fewer time cycles is needed to complete matrix-vector multiplication. Also, with more zero elements, the operation time becomes shorter, this is mainly because of reduced size of BILF files.

zero elements	zero rate	time cycles	BILF size(kb)	execution time(s)
6	9.38%	15	4630	0.71
13	20.31%	13	3801	0.41
19	29.69%	12	3367	0.6
25	39.06%	10	2679	0.28
31	48.44%	9	2317	0.37
38	59.38%	7	1718	0.20
44	68.75%	5	1177	0.17
49	76.56%	4	912	0.05
57	89.06%	2	429	0.02

table 3 zero element rate experiment result

Experiment of concentration of zero elements is also conducted. parameter x, y, t, c and zero element rate are kept constant, and concentration of zero elements is changed over the experiment. The result indicates that with higher concentration of zero elements, the operation time tends to be longer.

5. Conclusion

In this research, we have proposed automatic generation mapping solution for matrix-vector multiplication on the ring-connected architecture to realize parallel computing.

The proposed method is formulation of matrix-vector multiplication on defined ring-connected architecture, transform it into a SAT problem and use SAT solver to solve it. Details of ring-connected architecture and formulation are introduced.

In the experiment, times to generate mapping solution under different circumstances with different parameters are measured and examples of mapping are presented. Mapping solution of 16x16 matrix can be generated in short time, and with modified formulation, sparse matrix can be mapped in fewer time cycles.

Reference

1) Osawa, K., Sekiya, A., Naganuma, H., & Yokota, R. (2017, July). Accelerating matrix multiplication in deep learning by using low-rank approximation. In 2017 International Conference on High Performance Computing & Simulation (HPCS) (pp. 186-192). IEEE.

2) Qiao, Y., Shen, J., Xiao, T., Yang, Q., Wen, M., & Zhang, C. (2017). FPGA‐accelerated deep convolutional neural networks for high throughput and energy efficiency. Concurrency and Computation: Practice and Experience, 29(20), e3850.

3) https://www.maxeler.com/products/mpc-cseries/

4) https://software.intel.com/en-us/articles/intel-xeon-phicoprocessor-codename-knights-corner

5) https://software.intel.com/en-us/articles/quick-startguide-for-the-intel-xeon-phi-processor-x200-product-family

6) Berkeley Logic Synthesis and Verification Group, ABC: A System for Verification, Sequential Synthesis Release 80911. http://www.eecs.berkeley.edu/~alanmi/abc/

7) University of California, Brekeley, "Berkeley logic interchange format (BLIF)," 2005

8) Bell, N., & Garland, M. (2008). Efficient sparse matrix-vector multiplication on CUDA (Vol. 2, No. 5). Nvidia Technical Report NVR-2008-004, Nvidia Corporation.

9) Bell, N., & Garland, M. (2009, November). Implementing sparse matrix-vector multiplication on throughput-oriented processors. In Proceedings of the conference on high performance computing networking, storage and analysis (p. 18). ACM.

10) Xia, L., Gu, P., Li, B., Tang, T., Yin, X., Huangfu, W., ... & Yang, H. (2016). Technological exploration of RRAM crossbar array for matrix-vector multiplication. Journal of Computer Science and Technology, 31(1), 3-19.

11) Liu, W., & Vinter, B. (2015). Speculative segmented sum for sparse matrix-vector multiplication on heterogeneous processors. Parallel Computing, 49, 179-193.

12) Gharehbaghi, A. M., Maruoka, T., & Fujita, M. (2018, September). A New Reconfigurable Architecture with Applications to IoT and Mobile Computing. In IFIP International Internet of Things Conference (pp. 133-146). Springer, Cham.

13) Mathews, M., & Abraham, J. P. (2016, August). Automatic code parallelization with openmp task constructs. In 2016 International Conference on Information Science (ICIS) (pp. 233-238). IEEE.