IPSJ Transactions on System LSI Design Methodology Vol.13 31–34 (Feb. 2020)

[DOI: 10.2197/ipsjtsldm.13.31]

Short Paper

Synthesis and Generalization of Parallel Algorithm for Matrix-vector Multiplication

Yukio Miyasaka[1,a)] Akihiro Goda[1] Ashish Mittal[2] Masahiro Fujita[1]

Received: June 5, 2019, Revised: August 30, 2019,
Accepted: October 27, 2019

Abstract: Recently, there have been more chances to calculate matrix-vector multiplication due to the growing use of the neural network. We have proposed the method to automatically synthesize the optimum parallel algorithm for the given environment and synthesized an algorithm for matrix-vector multiplication of a specific size matrix with 4 nodes connected in a oneway ring. This paper proposes a method to generalize the synthesized algorithm to deal with any size matrix. We generalized the synthesized algorithm for the 32×32 matrix to calculate $N \times N$ matrix-vector multiplication.

Keywords: partial synthesis, program synthesis, automatic parallelization

1. Introduction

Matrix-vector multiplication (MVM) is a time-consuming calculation in a neural network. It requires many Multiply and Accumulate (MAC) operations. As each multiplication is independent, there has been much effort to parallelize MVM [1], [2], [3].

Some research has been conducted to parallelize designs automatically [4], [5], [6], [7]. They partitioned a dataflow into several blocks and distribute them among nodes considering data dependency and communication. We expect that we can derive a more efficient parallel design by transforming the dataflow according to the computing environment before partitioning it.

We have proposed the method to automatically synthesize the optimum algorithm in the given parallel environment from the specified input-output relation [8]. We have applied our automatic synthesis method to MVM to derive the algorithm executable with nodes (processing units) connected in oneway-ring-topology. We were able to synthesize the optimum algorithm for the 32×32 matrix with 4 nodes by analyzing the algorithms synthesized for smaller sizes of matrices.

This paper proposes a method to generalize the synthesized algorithm to obtain the algorithm for an arbitrary size matrix.

This paper is organized as follows: Section 2 explains partial synthesis, which is fundamental to our work. Section 3 describes our previous work on the synthesis of the optimum algorithm. Section 4 explains our method to generalize the algorithm. Section 5 discusses the effectiveness of the proposed method, and Section 6 concludes the paper.

[1] The University of Tokyo, Bunkyo, Tokyo 113–0032, Japan
[2] Indian Institute of Technology Bombay, Powai, Mumbai, Maharashtra 400076, India
[a)] miyasaka@cad.t.u-tokyo.ac.jp

2. Partial Synthesis

Partial synthesis generates a new design equivalent to the specification by filling the blanks in the given template. The specification is a correct design or a set of correct input-output patterns. The template is an incomplete design, which has several blanks inside where each blank has candidates to fill itself.

We can formulate the synthesis problem as a 2-level Quantified Boolean Formula (2QBF) shown in Eq. (1). The formula is satisfied when there exists the assignment of candidates for all blanks (y) satisfying the following condition: the output pattern of the specification ($SPEC$) is equal to that of Template ($TMPL$) for every input pattern (x). This 2QBF can be solved by iteratively solving SAT problems [9]. The synthesis is successful if Eq. (1) is satisfied.

$$\exists y. \forall x. SPEC(x) = TMPL(x, y) \tag{1}$$

3. Synthesis of Parallel Algorithm

We modeled parallel environments as **Fig. 1**. Each node has several registers and a processor with a fixed number of inputs. Each register stores one variable. All nodes are synchronized and any operation in a processor takes one cycle. Nodes are connected

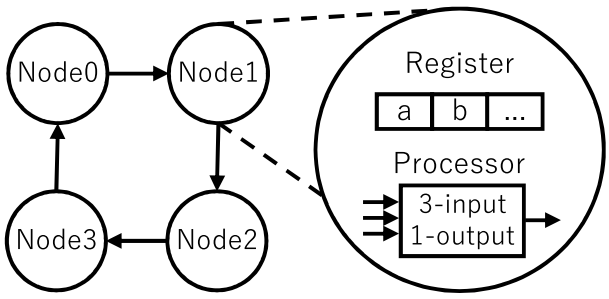


Fig. 1 The model of a parallel environment.

by oneway connections. The communication of one variable by a connection takes one cycle overlapped with operation. The modeled environment can be realized as a Coarse Grained Reconfigurable Array (CGRA), which works at 100–400 MHz when implemented on FPGA [10].

Our previous method creates templates as **Fig. 2** based on the model of a parallel environment. We denote a blank by {} with its candidates inside. A function beginning from 'f' is a blank function, which is an arbitrary function to be determined by the synthesis.

We first try synthesis with the template which takes just one cycle. If it fails, we try again with a new template with the number of cycles increased by one until the synthesis succeeds. The synthesized program takes the minimum possible number of cycles because SAT solver implicitly proves that there is no correct program with that number of cycles when the synthesis fails.

We performed the synthesis of MVM for $N \times N$ matrix with M nodes connected in oneway-ring-topology. We were able to synthesize a program for $N = M = 2$ but not $N > 2$ and $M \geq 2$ with a timeout of 1 day. To synthesize programs for larger matrices, we devised additional constraints as follows:

- Each processor performs only MAC operation
- N is dividable by M
- Nodes communicate only input-vector-elements
- The movement of data in registers is symmetric among nodes
- The movement of data is repeated every N cycles
- The movement of data is repeated every M cycles except the $i \times N$-th cycle where i is an integer

```
//cycle0
  //node0
    reg0_c0n0 = { inputs of program }
    reg1_c0n0 = { inputs of program }
    ... //repeat for the other registers in node0
    ope_c0n0 = f_c0n0(reg0_c0n0, reg1_c0n0, reg2_c0n0)
  //node1
    reg0_c0n1 = { inputs of program }
    ... //repeat for the other registers in node1
    ope_c0n1 = f_c0n1(reg0_c0n1, reg1_c0n1, reg2_c0n1)
  ... //repeat for the other nodes
  //communication by oneway connections
    con0_c0 = { reg0_c0n0, reg1_c0n0, ... }
    con1_c0 = { reg0_c0n1, reg1_c0n1, ... }
    ... //repeat for the other connections
//cycle1
  //node0
    reg0_c1n0 = { reg0_c0n0, reg1_c0n0, ...,
                  con3_c0, ...,
                  ope_c0n0 }
    ... //repeat for the other registers in node0
    ope_c1n0 = f_c1n0(reg0_c1n0, reg1_c1n0, reg2_c1n0)
  ... //repeat for the other nodes
  //communication by oneway connections
    con0_c1 = { reg0_c1n0, reg1_c1n0, ... }
    ... //repeat for the other connections
... //repeat for the other cycles
return { reg and ope at the last cycle }
```

Fig. 2 The template for a modeled parallel environment.

- The candidates for some blanks are reduced

We synthesized a program for $N = 16, M = 4$ in 22.5 seconds and one for $N = 32, M = 4$ in 3 hours using the same additional constraints. The synthesized programs finish in the minimum possible number of cycles, N^2/M, where we must do N^2 multiplications with M nodes doing MAC operation.

4. Generalization of Algorithm

The synthesis method explained in Section 3 seems impossible to synthesize a program for a larger matrix such as $N = 1000, M = 4$ in a feasible time. The synthesis time increases exponentially according to N as we can see from the previous results. It is reasonable because the synthesis problem is originally one of PSPACE problems and the size of problem increases according to N.

A way to compute MVM for $N = 1000, M = 4$ is to divide the matrix into 15625 sub-matrices of 8×8 and execute the program synthesized for $N = 8$ and $M = 4$ for each sub-matrix. However, it is not obvious whether the communication between each execution of the program can be completely hidden or not. If not, the entire calculation will take extra time for communication.

We propose a method to generalize the synthesized program. We use the generalization-template in **Fig. 3**. According to the constraints used in the synthesis, it has four loops where the number of iterations is N/M for two loops and M for the other loops. A matrix-element used in node p at cycle $k + j \times M^2 + i \times M \times N$ is defined by X and Y: X represents the row of the element, and Y represents the column of the element. They are defined with blanks s and c. The candidates of blank s are N, M, 1, 0, $-N$, $-M$, and -1. The candidates of blank c are N, M, and INF which is such a large int that x%INF is x for any int x. The number of lines for X is the same as that for Y. It is initially 1 and is increased by one when the generalization fails.

We synthesized X and Y with assigning 32 to N and 4 to M such that the generalization-template is equivalent to the program synthesized in our previous work for $N = 32, M = 4$. X and Y were synthesized as Eq. (2) in 1 second. The generalized algorithm takes N^2/M cycles, the minimum possible number of cycles.

$$\begin{aligned} X &= -M * i + p + N - M \\ Y &= -M * j + N - M + (-k + p + N)\%M \end{aligned} \tag{2}$$

```
for( int i = 0; i < N/M; i++) {
  for( int j = 0; j < N/M; j++) {
    for( int k = 0; k < M; k++) {
      for( int p = 0; p < M; p++) {
        // {s} == {N, M, 1, 0, -N, -M, -1}
        // {c} == {N, M, INF}
        X = ({s}*i + {s}*j + {s}*k + {s}*p + {s}) % {c}
          + ({s}*i + {s}*j + {s}*k + {s}*p + {s}) % {c}
          ...
        Y = ({s}*i + {s}*j + {s}*k + {s}*p + {s}) % {c}
          + ({s}*i + {s}*j + {s}*k + {s}*p + {s}) % {c}
          ...
        //out_vec[X] += matrix[X][Y]*vec[Y]
}}}}
```

Fig. 3 The generalization-template.

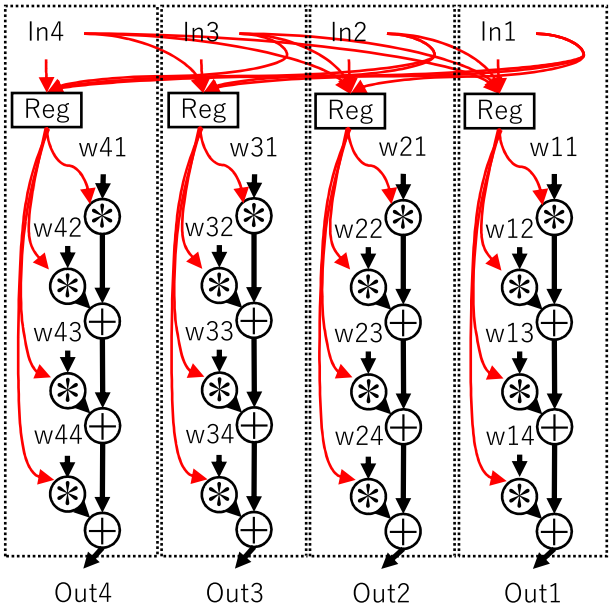


Fig. 4 The model of a parallel environment.

As of now, we do not have a method to prove the generalized algorithm is still correct when we change N and M to any number. When we use the generalized program, we should verify it after assigning specific values to N and M.

The verification consists of 2 parts: whether the outputs are correct and whether the communication is valid in the environment. We verify the first part by checking that each node uses all matrix-weights involved in the output-vector-elements the node outputs. For the second part, we adopted the assumption that a node must use the received input-vector-element in the next MAC operation whenever it receives. Then, we can verify the second part by checking that the input-vector-element used in a node was the most recently used in its adjacent nodes. The verification of Eq. (2) took 0.1 seconds for the first part and 5 seconds for the second part when we assign $N = 1000, M = 4$.

This template of generalization is different from a tiling method, although they look similar. A tiling method exploits the cache reuse. Our method generalizes the synthesized MVM program based on the constraints imposed in the synthesis.

5. Effectiveness

The synthesized algorithm takes the minimum number of cycles, which is proved through iterative synthesis. It must be faster or no less faster than what is generated by existing approaches.

We can see the effectiveness of the synthesized algorithm by comparing it with the straight forward algorithm shown in **Fig. 4** when $N = M = 4$. It is assumed that loading from external storage takes much longer time than communication among nodes. The straight forward algorithm requires 3 more registers for each node and takes 7 cycles in total including 3 cycles to broadcast vector-elements before MAC operation, whereas the synthesized algorithm reuses registers and takes only 4 cycles by overlapping the communication with MAC operation. When $N = 1000, M = 4$, the algorithm takes $N^2/M = 2.5 \times 10^5$ cycles. If CGRA works at 100 MHz, it requires 2.5 milli seconds.

6. Conclusion

We proposed the method to generalize the result of the synthesis method for the parallel MVM algorithm. We assumed that the algorithm can be represented by several loops with their number of iterations defined by the size of matrix and the number of nodes. We were able to generalize the program with those loops. Its correctness can be verified after the size of matrix and the number of nodes is defined.

References

[1] Kelefouras, V., Kritikakou, A., Papadima, E. and Goutis, C.: A methodology for speeding up matrix vector multiplication for single/multi-core architectures, *The Journal of Supercomputing*, Vol.71, No.7, pp.2644–2667 (online), DOI: 10.1007/s11227-015-1409-9 (2015).

[2] Hendrickson, B., Leland, R. and Plimpton, S.: An Efficient Parallel Algorithm for Matrix-vector Multiplication, *International Journal of High Speed Computing*, Vol.7, No.1, pp.73–88 (online), DOI: 10.1142/S0129053395000051 (1995).

[3] Codenotti, B. and Puglisi, C.: Matrix-vector multiplication: Parallel algorithms and architectures, *Computers and Mathematics with Applications*, Vol.16, No.12, pp.1057–1063 (online), DOI: 10.1016/0898-1221(88)90262-3 (1988).

[4] Mathews, M. and Abraham, J.P.: Automatic Code Parallelization with OpenMP task constructs, *2016 International Conference on Information Science* (*ICIS*), pp.233–238, IEEE (online), DOI: 10.1109/INFOSCI.2016.7845333 (2016).

[5] Mi, P., Zhao, Z., Sheng, W. and He, W.: An automatic parallelizer for Coarse-Grained Reconfigurable processor, *2016 13th IEEE International Conference on Solid-State and Integrated Circuit Technology* (*ICSICT*), pp.215–217, IEEE (online), DOI: 10.1109/ICSICT.2016.7998880 (2016).

[6] Nasiri, E., Shaikh, J., Hahn Pereira, A. and Betz, V.: Multiple Dice Working as One: CAD Flows and Routing Architectures for Silicon Interposer FPGAs, *IEEE Trans. Very Large Scale Integration* (*VLSI*) *Systems*, Vol.24, No.5, pp.1821–1834 (online), DOI: 10.1109/TVLSI.2015.2478280 (2016).

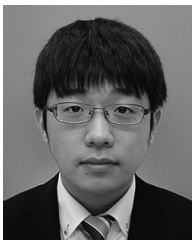
[7] Strauch, T.: Timing driven RTL-to-RTL partitioner for multi-FPGA systems, *2013 23rd International Conference on Field programmable Logic and Applications*, pp.1–4, IEEE (online), DOI: 10.1109/FPL.2013.6645579 (2013).

[8] Miyasaka, Y., Mittal, A. and Fujita, M.: Synthesis of Algorithm Considering Communication Structure of Distributed/Parallel Computing, *20th International Symposium on Quality Electronic Design* (*ISQED*), pp.45–51, IEEE (online), DOI: 10.1109/ISQED.2019.8697224 (2019).

[9] Solar-Lezama, A., Tancau, L., Bodik, R., Seshia, S. and Saraswat, V.: Combinatorial sketching for finite programs, *ACM SIGARCH Computer Architecture News*, Vol.34, No.5, p.404 (online), DOI: 10.1145/1168919.1168907 (2006).

[10] Taras, I. and Anderson, J.H.: Impact of FPGA Architecture on Area and Performance of CGRA Overlays, *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines* (*FCCM*), pp.87–95, IEEE (online), DOI: 10.1109/FCCM.2019.00022 (2019).

Yukio Miyasaka received his B.E. degree from the University of Tokyo in 2018. He is currently a Master's course student in the Department of Electrical Engineering and Information Systems at The University of Tokyo. His research interests include logic synthesis and formal methods.



Akihiro Goda received his B.E. degree from the University of Tokyo in 2018. He is currently a Master's course student in the Department of Electrical Engineering and Information Systems at The University of Tokyo. His research interests include logic synthesis and formal methods.

Ashish Mittal is a senior undergraduate student in the Department of Computer Science and Engineering at Indian Institute of Technology Bombay.

Masahiro Fujita received his Ph.D. degree in Engineering from the University of Tokyo in 1985 and shortly after joined Fujitsu Laboratories Ltd. From 1993 to 2000 he had been assigned to Fujitsu's US research office and directed the CAD research group. In March 2000, he joined the department of Electronic Engineering in the University of Tokyo as a professor, and now a professor at System Design Research Center (previously VLSI Design and Education Center) in the University of Tokyo. He has been involved in many research projects on various aspects of formal verification.

(Recommended by Associate Editor: *Nozomu Togawa*)