Technology Development of AI Application for In-vehicle Control ECU

Natsuki YOKOYAMA Yuu MORIYAMA

Abstract

Development of the AI technology in recent years is remarkable, and it is said that the AI products are no longer just a passing fad as they are becoming increasingly common even for general households.

We, DENSO TEN Limited, have also promoted the technology development for installing AI on an in-vehicle control ECU (Electric Control Unit) under such circumstances. However, AI is not supposed to be installed on an in-vehicle microcomputer at this time, and there are a number of issue toward the realization.

Usually, AI is being researched on the assumption that it will be installed on a parallel computing environment such as a GPU (Graphics Processing Unit), and also the AI development libraries which are widely used follow this policy.

Accordingly, the developed AI cannot be installed on the in-vehicle microcomputer as it is due to the processing load and architecture gap.

Therefore, we forcused on the fact that the AI structure was a network model, and that it could be interconverted with the block diagram used for in-vehicle code generation. In addition, a method was developed to connect microcomputer implementation and the AI design capable of being combined with MBD (Model Based Development) process, which had a proven track record in automotive control development.

1. Introduction

Approximately 60 years have passed since 1956 when the term “artificial intelligence (AI)” was proposed at the Dartmouth Conference, and the utilization of AI is spreading worldwide as a more practical and realistic technology. In this third AI boom which started around 2010, a number of practical systems have been realized, and unlike the past two AI booms which lost their exits and ended, it is expected not to end with a passing fad.

Successful case in various industrial fields are also being provided. For example, technology, which substitutes AI for specialized knowledge having difficulty being manualized or formulated simply, has achieved the performance equivalent to or better than that of human experts. Although its application fields cover a lot of areas such as medical, machine industry, electronic device, and architecture, AI shares a number of attributes with successful cases in terms of requirement of complicated and specialized knowledge even in the field of automobile control. Therefore, it is considered that the effects utilizing AI can greatly be expected.

We, DENSO TEN Limited, have also promoted the technology development for installing AI on a vehicle control ECU under such circumstances. However, AI is not supposed to be installed on an in-vehicle microcomputer at this time, and there are a number of issues toward the realization.

This article discusses the installation of AI on the in-vehicle microcomputer and the integration into MBD (Model Based Development) process which had a proven track record in automotive control development.

2. AI Installation Issue on In-vehicle Control ECU

As described above, AI is not supposed to be installed on the in-vehicle microcomputer mounted on the in-vehicle control ECU. Extremely high reliability

is required for the in-vehicle microcomputer under severe temperature, power consumption conditions, and the like. On the other hand, it doesn't have sufficient processing performance compared to GPU (Graphics Processing Unit) mainly used as an AI development environment. In addition, the architecture differs greatly from GPU specializing in parallel processing. Since many AI development environments are optimized for GPU, the created AI has a large gap in processing performance and architecture, making it difficult to be installed on the in-vehicle microcomputer.

And now, there are two following processes in AI; "leaning" which enables to obtain a desired output for a given input and "inference" which returns the output to the input based on the learning results.

AI which is currently considered to be installed on the in-vehicle microcomputer can perform "inference," and "leaning," which requires extremely high processing power, is reasonable to be implemented in the GPU server environment with plenty of resources, or others.

Many libraries are available for the GPU server environment, and if there is a certain level of knowledge, the AI development can be carried out. However, its deliverables are usually coded in Python, which are premised on cooperation with GPU and cannot be directly installed in the in-vehicle microcomputer environment. In order to be installed on the in-vehicle microcomputer, conversion to C language where a compiler exists is necessary in some way. Although several methods for converting Pythod code to C language have been proposed, the problem is far from a solution because Pythod code corresponding to the architecture of the in-vehicle microcomputer needs to be prepared.

Therefore, applying a block diagram with a proven track record in the MBD process is proposed. For MBD, the control design based on block diagram is performed with tools such as Simulink®. This block diagram has a function capable of simulation on PC and also capable of conversion to C language installable on the in-vehicle microcomputer. Thus, this enables the issues to be narrowed down to see whether AI can be expressed with a block diagram.

On the other hand, it is difficult to essentially solve the gap in processing performance. There is a profound difference in processing performance between GPU and the in-vehicle microcomputer in specifications, because the roles required for both are different. For that reason, AI which is developed on GPU server environment must be limited to an executable size by the in-vehicle microcomputer for inference. Nevertheless, since it is difficult to determine exactly how much processing performance is needed for AI before the installation, whether the processing load can be estimated in advance becomes a focal point.

3. Mutual Conversion between Network Model and Block Diagram

The method of representing AI on a block diagram, which is described in the previous chapter, is considered in this chapter. If AI which is discussed in this article is a deep learning, the structure is a network model. The network model is a weighted directed graph. In addition, a general block diagram can be regarded as a weighted directed graph if a block value is considered as weight, which means that network model can be converted to the block diagram **(Fig. 1)**. Furthermore, this conversion is reversible because no loss of information occurs and the reverse conversion from the block diagram to the network model is also possible.

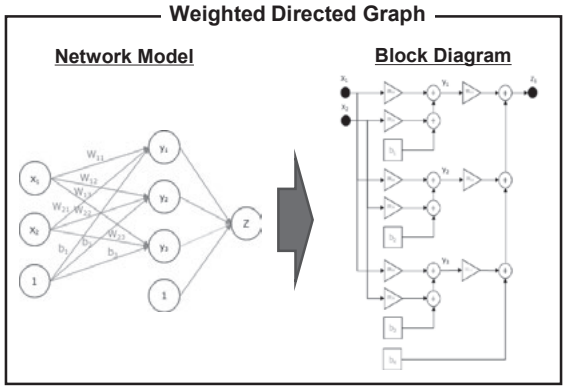

Fig. 1 Conversion from Network Model to Block Diagram

For example, the following network model is considered **(Fig. 2)**. x is an input node, y is an intermediate node, z is an output node, weight is represented by w, and bias is represented by b. c shall be a constant with 1 as a value.

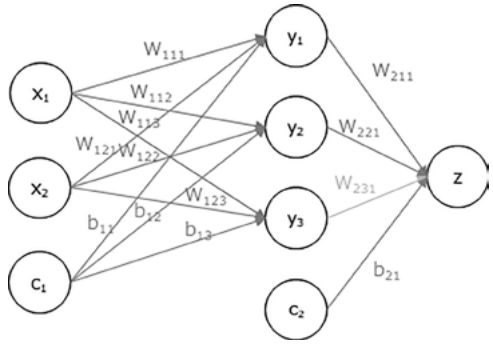

Fig. 2 Example of Network Model

If this is regarded as the weighted directed graph, the adjacency matrix can be represented as follows **(Table 1)**.

Table 1 Adjacency Matrix of Network Model

	x_1	x_2	c_1	y_1	y_2	y_3	c_2	z
x_1	0	0	0	w_{111}	w_{112}	w_{113}	0	0
x_2	0	0	0	w_{121}	w_{122}	w_{123}	0	0
c_1	0	0	0	b_{11}	b_{12}	b_{13}	0	0
y_1	0	0	0	0	0	0	0	w_{211}
y_2	0	0	0	0	0	0	0	w_{221}
y_3	0	0	0	0	0	0	0	w_{231}
c_2	0	0	0	0	0	0	0	b_{21}
z	0	0	0	0	0	0	0	0

If the determinant of a graph is created from the adjacency matrix, it can be represented as follows [Formula (1), Formula (2)].

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} w_{111} & w_{121} & b_{11} \\ w_{112} & w_{122} & b_{12} \\ w_{113} & w_{123} & b_{13} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ c_1 \end{bmatrix} \quad \cdots(1)$$

$$\mathrm{z} = \begin{bmatrix} w_{211} & w_{221} & w_{231} & b_{21} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ c_2 \end{bmatrix} \quad \cdots(2)$$

Here, if x, y, z, w, and b are represented respectively as vectors such as X, Y, Z, W, and B and plugged in 1 for c, they can be deformed as follows **[(3), (4)]**.

$$\mathrm{Y} = \begin{bmatrix} W_1 & B_1 \end{bmatrix} \begin{bmatrix} X \\ 1 \end{bmatrix} = W_1 X + B_1 \quad \cdots(3)$$

$$\mathrm{Z} = \begin{bmatrix} W_2 & B_2 \end{bmatrix} \begin{bmatrix} Y \\ 1 \end{bmatrix} = W_2 Y + B_2 \quad \cdots(4)$$

The above formulas are represented as the block diagram as below. Therefore, the network model can be converted into the block diagram **(Fig. 3)**.

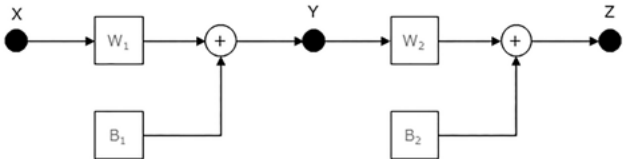

Fig. 3 Converted Block Diagram

Also, the block diagram has a function of abstracting the control structure by hierarchization. Usually, control designers are not interested in the internal structure of AI. For that reason, if AI is handled as a block with complex functions, the network model can be hidden. On the other hand, AI developers can proceed with development regardless of the control if they focus only on the hidden AI block. Furthermore, if the AI network structure is abstracted by the hierarchization such as an input layer, a convolution layer, and the like, its lower layers can be simplified to a combination of simple numerical calculations. This hierarchy is used by engineers installing software on the microcomputer to increase speed by devising parallelization, memory allocation, or the like. Thus, the block diagram conversion of the network model is not only to install AI on the in-vehicle microcomputer, but also to separate the development process by role, enabling the effect of clarifying the quality and performance guarantee ranges to be expected.

4. Weight Saving of Network Model and Restrictions on ECU Installation

As for AI, which was converted to a block diagram by the method described in the previous chapter, C language code which is installable on the in-vehicle microcomputer can be generated by the Simulink® function. However, just because of this, it doesn't mean that the installed program can fit in the executable processing performance. If a problem of processing performance is identified after execution, the impact of rework is great because it causes us to start over from AI design. Therefore, the method for estimating the required processing performance at the earliest possible stage is required.

Accordingly, we propose a method to formulate a standard calculation scale for each layer of the network model and also define its integral result as the required processing performance of AI. In addition to that, it is possible to have an estimate on which microcomputer the AI software in the design stage can be implemented by defining the processing performance for each microcomputer.

For example, the following calculation scale of the convolutional layer is considered **(Fig. 4)**. In the calculation of the convolution layer, a feature map is created by multiplying each element of the filter and adding them, which is slid by the size of input. Furthermore, a plurality of filters can be used.

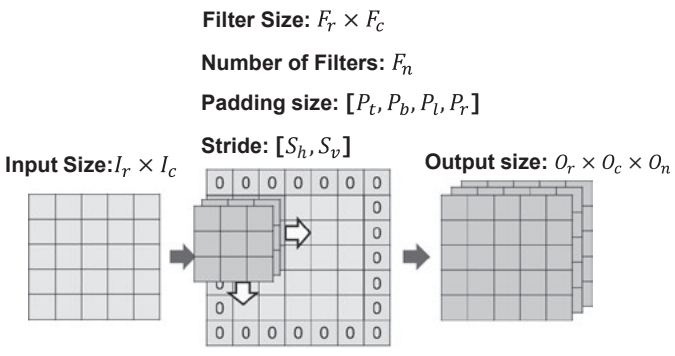

Fig. 4 Calculation Scale of Convolutional Layer

To put it simply, it is clearly the case that the calculation scale N is proportional to the input size $I_r \times I_c$, the filter size $F_r \times F_c$, and the number of filters F_n. Considering the padding $[P_t, P_b, P_l, P_r]$ which fills the outer periphery of the input with 0 and the stride $[S_h, S_v]$ at the time of sliding the filter, the following equitation (5) can be derived.

$$N = (F_r \times F_c) \times \frac{I_r + P_l + P_r}{S_h} \times \frac{I_c + P_t + P_b}{S_v} \times F_n \quad \cdots(5)$$

The point of this method is to be able to quantitatively explain which layer becomes a bottleneck, in addition to being able to estimate the required processing performance at the AI design stage. Because of a device-independent index, it can be easily used regardless of a target.

The following four main methods for reducing the AI calculation scale can be considered; Method ① Select a model suitable for an application. Method ② Narrow down the input and intermediate paths to those that have a high impact on the output. Method ③ Suppress the calculation accuracy to the minimum requirement. Method ④ Convert the GPU-based code into code according to the target.

As for method ①,

There is a method of transfer leaning from AI, which was used in the most recent previous case. However, confirming whether the calculation scale was optimized in the previous case is necessary.

As for method ②,

it can be further divided into two methods. One is to select an input capable of being expected to contribute to an output. Since knowledge of the target application is required for this, AI installation for ECU also become a point where the expertise as a supplier can be demonstrated. The other is known as sparse modeling by deleting paths with a small amount of weight.

As for method ③,

methods such as quantization have been thought to be effective. Although the data which is used in a neutral network is usually floating-point numbers of 16bit or 32bit, converting this into a fixed-point number of 8bit or into a truth value of 1bit in an extreme case enables the data to be lighter.

As for method ④,

the response varies depending on the speed-up function mounted on a device to be targeted. However, the key is usually how far calculations

without any dependency can be parallelized. Specifically, how efficient the inner products of a totally coupled layer and a convolution layer can be calculated, contributes to the weight saving.

The effect can be measured at the AI design stage by methods ① to ③ out of these four methods, which means that the formulation is possible with the aforementioned calculation scale. On the other hand, because the method ④ is a device-dependent performance, it needs to be considered individually. However, this can be resolved by preparing AI dedicated libraries referenced by code generated from the block diagram for each target. In other words, this ends up clearly being separated as an issue for engineers mounting a microcomputer.

5. Development of AI Dedicated Library

If AI which is designed/learned on the GPU server environment cannot structurally reduce the calculation any further, it can be narrowed down how to improve CPI (Clocks Per Instruction) as an issue when implementing AI in a microcomputer. Although various methods using a low latency command, memory access, or the like are known, parallelism contributes most to the performance. For that reason, considering how far the dependency relationship of calculation can be eliminated is necessary when developing libraries.

Which part of the calculation has the dependency relationship can be clearly understood if the structure of the block diagram before performing the code generation is regarded as a directed graph. By using this, we propose a method for examining parallelization graphically with the block diagram.

For example, the following block diagram is assumed. Since blocks which are connected by arrows clearly have the dependency relationship, a target to be parallelized is selected for each branch **(Fig. 5)**.

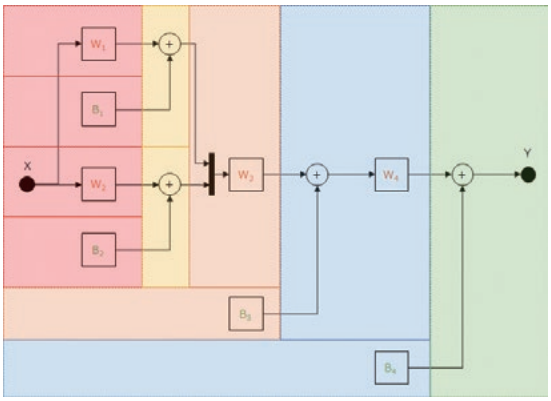

Fig. 5 Examination of Block Diagram Parallelization

A horizontal axis represents a time axis and a vertical axis represents the parallel number for divided areas, showing the maximum parallelization and the shortest time state at the time. If the parallel number of a target is sufficient, this structure is implemented as it is. If not, the allocation needs to be reviewed. Accordingly, this divided area is regarded as a directed graph and the allocation is performed in order of the end node **(Fig. 6)**.

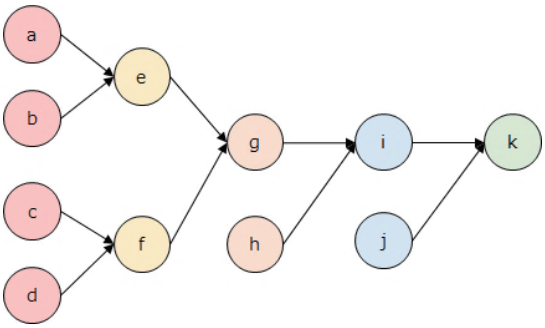

Fig. 6 Graphed Processing Dependency

Now, if the target is a dual-core CPU, the parallel number is 2. For the sake of clarity, assuming that all the processing from a to k have a similar processing time, we just have to prepare boxes multiplying the parallel number by time series and arrange the processing so that the time series is made as short as possible. The algorithm is as follows. Procedure ① Organize each node in a direction toward time series so that it shall be one node ahead of the other nodes being directly referenced by each node. Procedure ② The "rate being depended" which is the total number of nodes referring to each node is required. Procedure ③ Place the nodes to the far right of empty boxes in order of

the highest rate being depended. However, they shall be placed to the left of the position organized in procedure ① **(Fig. 7)**.

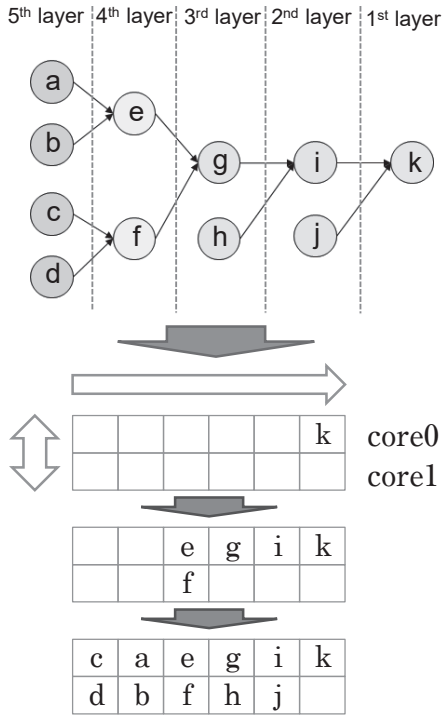

Fig. 7 Allocation Procedure for Parallel Processing

6. Conclusion

Creative efforts, which are described above, enabled the installation process to be established while taking the measure of the AI scale according to the target. This method is considered to be successful in terms of the fact that the mature MBD process can be applied even to a new item called AI as before.

We would like to perform development trials using this method and make continuous improvements toward the mass production development of ECU applying AI in future.

・Simulink® is a registered trademark of the MathWorks, Inc.

Profiles of Writers

Natsuki
YOKOYAMA

AE Business Group
Advanced System
Development Div
Engineering
Department

Yuu
MORIYAMA

AE Business Group
Advanced System
Development Div
Engineering
Department