Federated Learning Platform on Embedded Many-core Processor with Flower

Masahiro Hasumi and Takuya Azumi
Graduate School of Science and Engineering
Saitama University

***Abstract*—In the emerging field of autonomous vehicle development, the role of artificial intelligence, particularly deep learning (DL), has garnered significant interest. This growing interest has led to extensive studies in using cameras and other onboard sensors for the critical task of object detection and recognition in these vehicles. A major concern, however, is the sensitive nature of the training data, which poses privacy risks when centralized on a server. Furthermore, as the demand for privacy protection increases, power consumption may increase rapidly. To address these issues, this paper proposes Federated Learning (FL) for DL on an embedded many-core processor. This study provides an implementation of FL, aiming to enhance privacy protection and energy efficiency in autonomous vehicle development. The experimental results of the proposed FL platform demonstrate that it offers significant improvements in processing speed and power consumption, suggesting an enhanced performance compared to existing edge devices.**

***Index Terms*—Federated learning, many-core processor, deep neural network, embedded systems**

I. Introduction

The advent of autonomous vehicles [1] has revolutionized the concept of transportation, introducing a future where vehicles operate independently and efficiently. At the heart of this transformation lies the integration of advanced computing technologies, where each vehicle becomes a hub of automated decision-making. This paradigm shift towards autonomous vehicles underscores the essential role of sophisticated algorithms and computational systems, setting the stage for artificial intelligence (AI) to emerge as the backbone of autonomous vehicle technology.

AI, particularly deep learning (DL), stands at the forefront of both revolutionizing autonomous vehicle technology and streamlining manufacturing operations. However, the development of DL-based solutions often necessitates a considerable amount of data collected from various IoT devices, a process critical for training high-quality DL models. In the case of autonomous vehicles, managing and transmitting this data must be handled with extreme care, as it may contain sensitive privacy information. Here, the concept of federated learning (FL) [2] comes into play as a potential solution. FL enables the training of models without centralizing the data on a server, reducing the risk of privacy breaches during data transmission. In the FL paradigm, each device or node in a network trains a local model using its own data and then shares only the model updates with a central server. This decentralized approach not only safeguards privacy but also fosters collaborative model improvement. Thus, FL is an ideal approach for autonomous vehicles that need to be careful with their data.

Applying FL to autonomous vehicles presents specific challenges. These vehicles operate multiple systems simultaneously, including sensing and path planning, requiring a significant amount of computing resources. Moreover, the learning process consumes a considerable amount of power, which must be carefully managed for onboard systems. To tackle these challenges, many-core processors are becoming instrumental. Many-core processors allow for efficient parallelization of tasks, enabling the trade-off between high computing power and power-saving to be achieved simultaneously. This paper proposes a FL platform on many-core processors for autonomous driving systems. In the context of autonomous vehicles, these processors can distribute the workload efficiently, ensuring that the demands of FL model training do not compromise other critical vehicle functions.

In summary, the fusion of FL and many-core processors offers a promising path for enhancing the capabilities of autonomous vehicles. This technological synergy not only addresses data privacy concerns but also optimizes the computational aspects of AI, thereby contributing to the advancement of autonomous driving technologies.

The main contributions of this paper are summarized as follows:

- We define and implement training of convolutional neural networks (CNNs) trained in Federated Learning on a clustered many-core processor environment.
- We compare training speeds and power consumption in a scalable many-core environment, shedding light on its efficiencies.
- We compare performance with common embedded devices and show the usefulness of many-core processors.

The remainder of this paper is organized as follows. In Section II, we provide a detailed description of FL and many-core processors. In Section III, we explain the proposed framework. In Section IV, we evaluate the proposed framework, and in Section V, we discuss related work. Finally, a brief conclusion is provided in Section VI.

II. System Model

This section provides the system model of this research with relevance to vehicular IoT, as shown in Fig. 1. This study assumes the use of many-core processors and FL. The FL framework is applied to the entire system, and the DL

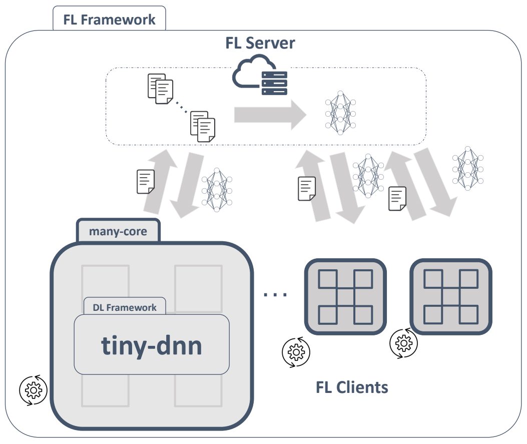


Fig. 1. System Model.

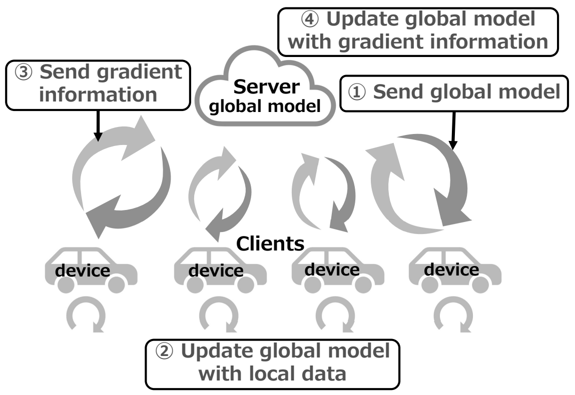

④ Update global model with gradient information
③ Send gradient information
Server global model
① Send global model
Clients
device
device
device
device
② Update global model with local data

Fig. 2. Federated Learning Process.

framework running on a many-core processor is assumed to be the client. Section II-A describes FL, whereas Section II-D describes the many-core processor, the target platform of this study.

A. Federated Learning

Traditional machine learning algorithms often follow a centralized learning paradigm in which training data held by the client are aggregated on a server, and the server uses the data to learn and create a model. In centralized learning, supercomputers with large resources are used to aggregate a large amount of data to enable rapid learning; however, centralized learning has a major limitation. By consolidating the training data in one place, the data can be leaked in bulk if the server gets hacked. As information obtained from devices such as smartphones, may contain personal information, consolidating the information obtained from a single server is risky. FL is used to eliminate the risks associated with aggregation. In FL, the server has a global model, and each client has a local model. In centralized ML, the server performs the training, whereas in FL, each client performs the training. The server sends the global model to the clients, and each client trains using its own training data. The clients then send their updated models to the server, which aggregates the models and updates the global model. Again, the server repeats the cycle of sending the updated global model to the client. In this way, the server can update the global model without aggregating the training data of the clients. Since the training data is stored locally by each client and trained by the local model, the risk of information leakage is reduced. The above flow is shown in Fig. 2. Although multiple studies have been conducted on model aggregation methods on servers, this study uses FedAvg [2], which is a representative method. FedAvg considers the arithmetic mean of the weights sent by each client and updates the global model.

B. Federated Learning Framework

In this study, Flower [3] is used as the FL framework. Flower is a friendly framework for collaborative AI and data science. FL is difficult to implement in terms of scale and system heterogeneity realistically. A number of research frameworks exist to simulate FL algorithms. However, FL algorithms do not address the study of scalable FL workloads in heterogeneous edge devices. Flower makes new approaches, such as collaborative learning and collaborative evaluation, available to a wide range of researchers and engineers. The server specifies the number of rounds, model size, initial values, aggregation, and server address. The client specifies a machine learning model and performs training. For communication between the server and client, gRPC is used to send and receive weight and gradient information. With Flower, developers can build FL systems without having to coordinate communications.

C. Deep Learning Framework

In this study, tiny-dnn [4] is chosen as the framework for DL. Important reasons exist for this choice. Commonly known DL frameworks, such as Tensorflow [5] primarily target Python and feature rich Python libraries and community support. However, this study targets embedded systems and uses C/C++ as the development language. This makes it difficult to use a Python-based framework directly. tiny-dnn is a DL framework designed to support C++ development and is provided in the form of header files. This characteristic allows it to be used on clustered many-core processors and provides excellent portability since it does not involve library downloads or dependency issues.

Thus, the use of tiny-dnn is an appropriate choice to facilitate DL implementation in C/C++-based embedded systems.

D. Many-core Processor

Many-core processors enable high computing power and power savings by utilizing a large number of cores. The many-core architecture called MPPA3-80 (Coolidge) was developed by Kalray over three generations [6]–[8]. Each computation unit of Coolidge architecture is known as a compute cluster, and many-core processors composed of compute clusters are known as clustered many-core processors [6].

Coolidge has five compute clusters and adopts a Network-on-Chip (NoC) structure with multiple paths to prevent bus

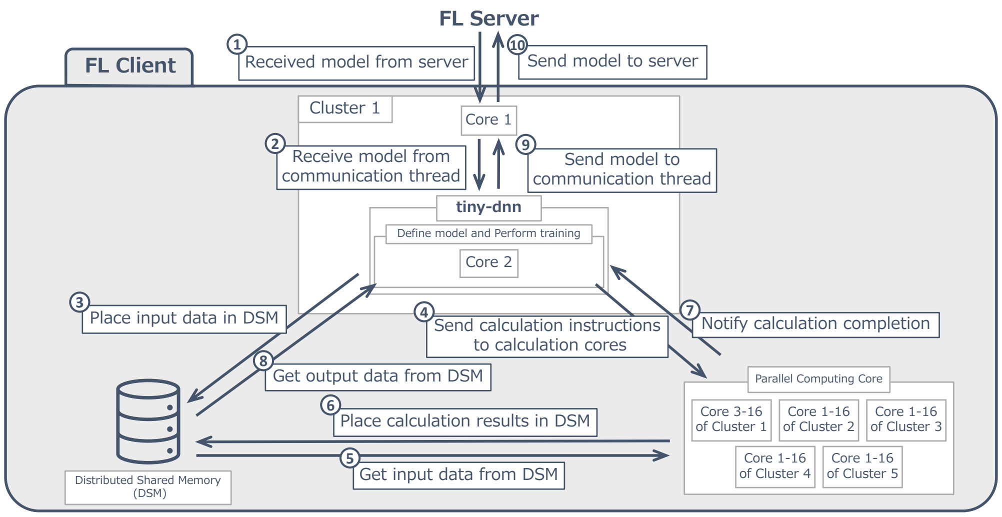

FL Server
FL Client
① Received model from server
⑩ Send model to server
Cluster 1
Core 1
② Receive model from communication thread
⑨ Send model to communication thread
tiny-dnn
Define model and Perform training
Core 2
③ Place input data in DSM
④ Send calculation instructions to calculation cores
⑦ Notify calculation completion
⑧ Get output data from DSM
Parallel Computing Core
Core 3-16 of Cluster 1
Core 1-16 of Cluster 2
Core 1-16 of Cluster 3
Core 1-16 of Cluster 4
Core 1-16 of Cluster 5
⑥ Place calculation results in DSM
⑤ Get input data from DSM
Distributed Shared Memory (DSM)

Fig. 3. Process Flow of Proposed Framework.

contention among the clusters [7]. Each compute cluster has 16 compute cores. Each compute cluster shares 4 MB of local memory for inter-cluster communication. Coolidge can run standard Portable Operating System Interface multi-threaded C/C++ applications. As Coolidge has additional cores than a normal multi-core processor, the clock frequency per core can be reduced, thus improving power efficiency. With improved power efficiency, Coolidge achieves low power consumption while maintaining high performance. Furthermore, as each compute cluster can execute independent applications, multiple applications can be executed in parallel on the Coolidge platform. Therefore, the overhead is reduced compared to conventional embedded devices, and meeting the high level of real-time performance of autonomous driving systems is possible.

III. APPROACH

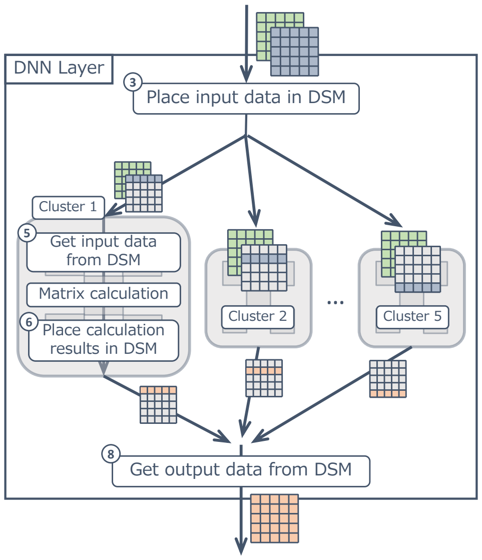
In this section, we describe in detail the new approach in our research. In traditional machine learning methods, the learning process was executed primarily on a powerful server, which required high computational power. In contrast, FL disperses the learning process across individual client devices. This distribution presents unique challenges as each device must now provide its own computational power, thereby increasing the power consumption of the client device. In particular, in the environment of embedded devices such as autonomous vehicles, these devices are simultaneously running more than 50 applications such as sensor data processing, localization, path planning, with limited computational resources. Given these constraints, it would be impractical that additional computational power from client devices would not be appropriate for FL.

To address the issue, this study proposes a novel approach that utilizes clustered many-core processors within client devices. Clustered many-core processors have multiple cores and can perform parallel processing, making them suitable for environments where many applications are processed simultaneously, such as autonomous vehicles. In addition, the matrix computations fundamental to DL are naturally suited for parallel execution across these cores, which complements the capabilities of clustered many-core processors. Details of the parallelization of the computation are presented in Section III-B. The utilization of clustered many-core processors facilitates distributed processing and lower clock frequencies, thereby reducing the power consumption of the learning process on the client device. This approach has the potential to improve energy efficiency and enhance the sustainability of FL execution in embedded devices.

We aim to materialize the proposed architecture and conduct experiments to evaluate its performance and usefulness precisely. By forging ahead in this domain, our work will open up new possibilities by contributing to the evolution of machine learning on embedded devices and will provide a valuable foundation for future research.

A. *Process Flow*

The process flow is shown in Fig. 3. In this figure, only one client is shown for illustrative purposes. The process consists of ten major steps and proceeds as follows. Core 1 of Cluster 1 receives the model from the server (Step 1) and applies it to the model of the tiny-dnn thread in Core 2 (Step 2). tiny-dnn places the data in distributed shared memory (DSM) during the computation of the layers of the CNN (Step 3) and sends computation instructions to the compute cores (Step 4). DSM is a memory shared among clusters. The computation core copies the data from the DSM and performs a parallel computation (Step 5). Details of this parallel computation are given in Section III-B. The results of the computation are placed in the DSM (Step 6), and the tiny-dnn thread is notified of the end of the computation (Step 7). The tiny-dnn thread retrieves the output data from the DSM (Step 8) and sends the model to the server via Core 1 (Steps 9 and 10).



DNN Layer
3 Place input data in DSM
Cluster 1
5 Get input data from DSM
Matrix calculation
6 Place calculation results in DSM
Cluster 2
...
Cluster 5
8 Get output data from DSM

Fig. 4. Parallel Calculation.

Fig. 5. Traffic Sign Image.

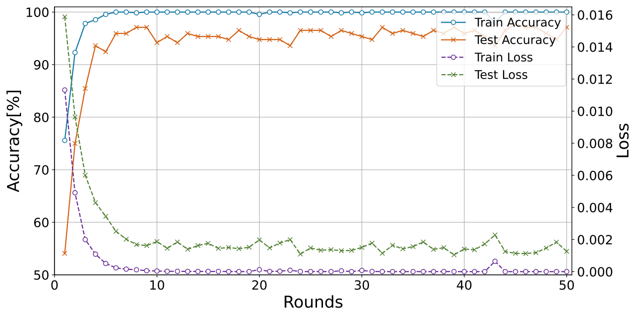


Fig. 6. Accuracy and Loss.

B. Parallelization Approach

The parallelization approach to computation on clustered many-core processors is deployed in the following sequence of steps to achieve a high degree of computational performance and promote optimal use of computational resources. Details of the parallelization between clusters are shown in Fig. 4. The numbers in the figure correspond to each step in Fig. 3. In the parallelization section, the tiny-dnn thread places the input data in the DSM and assigns the part of the matrix computation to the computation cluster. Each compute cluster retrieves data from the DSM and starts computing. Within the cluster, each core distributes computation tasks according to its instructions, performs the computation, and writes the computation results to the DSM. The tiny-dnn thread retrieves the computation results from the DSM, integrates them, and produces the final output.

IV. EVALUATION

This section focuses on evaluating the performance of FL on clustered many-core processors. The content related to the evaluation is divided into the following sections. Section IV-A describes in detail the training data sets and models used in the evaluation. Since these elements form the basis of the evaluation, information on their selection and configuration is provided.

The following evaluation items will be conducted in this study:

- Accuracy and Loss: We examine the performance of FL using clustered many-core processors and evaluate its usefulness.
- Scalability: We evaluate the change in learning time and power consumption with different numbers of cores.
- Device Comparison: We compare and evaluate the main embedded devices and their processing speeds.

A. Dataset and DNN Model

The German Traffic Sign Recognition Benchmark (GTSRB) [9] was selected for evaluation in this study. The GTSRB is a detailed dataset of German traffic signs, which contains 43 different classes of traffic signs. The dataset is widely used in road sign identification tasks and contributes to important applications in the field of autonomous vehicles and road safety. The GTSRB contains road sign images of different sizes and brightness. When it comes to road sign recognition for autonomous vehicles, high accuracy and reliability are required. The GTSRB is, therefore, an ideal dataset for evaluating models in situations close to real-world scenarios. Some examples of road signs in the GTSRB, featuring different shapes and colors, are depicted in Fig. 5.

In addition, a lightweight and effective road sign identification model, "DeepThin" [10], was employed in this study. This model does not require a GPU and features high performance in resource-limited environments. In embedded systems such as autonomous vehicles, resource efficiency is important and the use of models such as DeepThin offers significant advantages. This study uses cross entropy, which is used in classification tasks, as the loss function.

B. Accuracy and Loss

First, we show that the training is performed accurately on a many-core processor. The number of mini-batches is set to 32, with ten epochs as one round, for a total of 50 rounds of training. We evaluate the learning progress by measuring accuracy and loss after each round of aggregation. The training data used are selected from among 860 data sets, 20 for each class, of which 688 (80%) are used as train data and the remaining 20% are used as test data. The

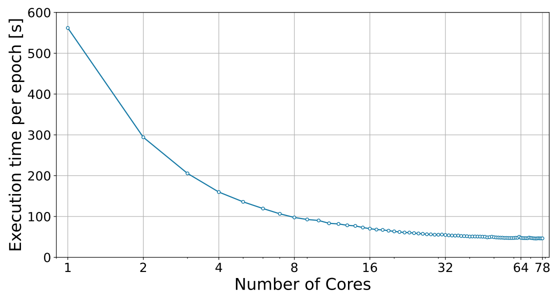


Fig. 7. Execution Time and Power Consumption.

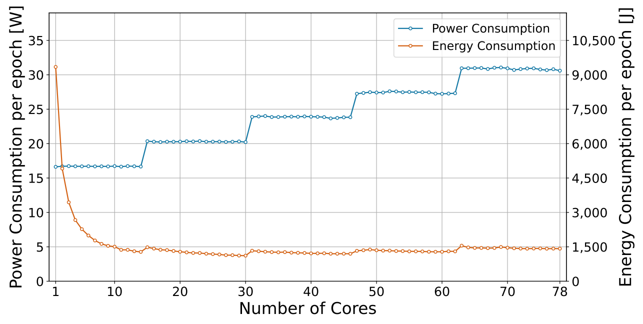


Fig. 8. Power and Energy Consumption.

results of the accuracy and loss measurements are shown in Fig. 6. From this figure, it can be seen that accuracy improves, and loss decreases as training progresses. These results indicate that learning on the many-core processor is effective and that an appropriate learning process is in place. Furthermore, these results support the ability of many-core processors to efficiently handle complex machine learning tasks and highlight the effectiveness of learning algorithms in many-core environments.

C. Scalability

We provide a comprehensive analysis of the advantages of many-core processors in evaluating scalability. One of the features of many-core processors is that the number of cores used can be flexibly specified and that highly parallel processing is possible. In this evaluation, as in Section IV-B, the number of mini-batches was set to 32, and 688 images (80% of the 860 image data set) were used as training data, and the execution time and power consumption per epoch were measured.

Core 1 of Cluster 1 is dedicated to communication between the FL server and the client, and therefore, it does not contribute to the computation. Core 2 of Cluster 1 does not contribute to the computation in the same way as Core 1, as it defines the model and distributes the parallelism. As a result, up to 78 cores are available for parallel execution. The change in execution time as the number of cores changes is shown in Fig. 7. The graph of execution time uses a logarithmic scale on the horizontal axis. This figure shows a significant decrease in execution time as the number of cores increases, suggesting the high computational efficiency of many-core processors. This decrease in runtime underscores the importance of maximizing the processor's computational power, especially for large-scale machine learning tasks.

The changes in power consumption and energy consumption with changes in the number of cores are shown in Fig. 8. As for power consumption, the vertical axis of the graph shows the average power consumption during the training runs. Power consumption tends to increase as the number of clusters used increases. This is because cores not used for computation also run the operating system and consume a certain amount of power even in standby mode. However, the power-saving effect is realized within the same cluster by distributing the processing. Therefore, when using the same number of clusters, the power consumption does not significantly increase.

It is worth noting that the power consumption baseline increases with the number of clusters. However, the rate of increase in power consumption is small compared to the rate of decrease in execution time. Thus, the energy consumption shows a decreasing trend as the number of cores increases, as shown in Fig. 8. Although the percentage decrease in energy consumption becomes smaller as the number of cores increases, the increase in the number of cores decreases the time required for learning, indicating that the machine learning process can be completed faster with equivalent energy consumption.

The above analysis shows that the use of many cores in many-core processors combines high computational power with power savings. This means that many-core processors are highly scalable and can operate efficiently in a variety of large-scale machine learning tasks.

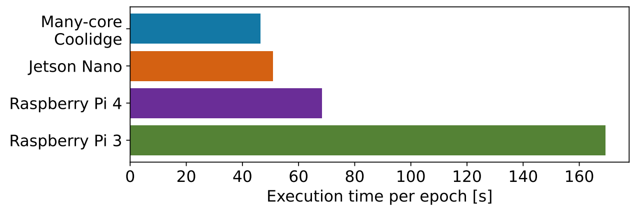


Fig. 9. Compare Embedded Devices.

D. Device Comparison

The following evaluation compares embedded devices. The evaluation targets are the Raspberry Pi 3 Model B+, Raspberry Pi 4, and NVIDIA Jetson Nano. We measured the execution time of one epoch with the same model and training data, and the comparison results are shown in Fig. 9. Many-core processor is faster than the NVIDIA Jetson Nano, which has a GPU, even though the many-core processor has only a CPU. These results are useful in the area of embedded systems, where many-core processors have the ability to perform faster computations.

TABLE I
COMPARISON OF THE PROPOSED PLATFORM WITH RELATED STUDIES

	FL	*Training Task*	*Many-core processors*	*Real time OS*
ROS-lite [7]			✓	✓
Many-core in DL inference [8]			✓	✓
Many-core in DL training [11]		✓	✓	✓
FedIoT [12]	✓	✓		
FL in Autonomous Vehicle [13]	✓	✓		
FL for LiDAR [14]	✓	✓		
This paper	✓	✓	✓	✓

V. Related Work

This section introduces the existing studies and compares them with this study in Table. I.

ROS-lite [7] mentioned the basic evaluation of many-core platforms in embedded systems. ROS-lite is a Robot Operating System (ROS) development framework for many-core platforms using NoC technology. This framework is characterized by low memory consumption and achieves communication between components provided by ROS running on each core on a multi/manycore platform.

Many-core in DL inference [8] evaluates the inference performance of DL using many-core processors. YOLOv3 is used as the evaluation model. The paper shows that the use of many-core processors enables efficient parallel computation and fast inference. However, this paper only evaluates inference using many-core processors and does not mention learning.

Many-core in DL training [11] evaluates the learning performance of DL using many-core processors. It is a foundational study of this research and shows that the use of many-core processors in training enables efficient parallel computation and fast learning.

To realize FL in real devices, a FedIoT platform Ref. [12] with a simple and effective design was proposed by Zhang et al. The FedIoT platform is divided into an application layer, an algorithm layer, and an infrastructure layer; thus, FL using the real device is facilitated.

FL in autonomous vehicles [13] is researching a system for dynamically allocating resources required for communication and learning in FL on embedded systems such as autonomous vehicles. Creating high-quality models demands substantial computational resources; in parallel, FL necessitates communication with a server, thus requiring resource allocation to enhance the quality of this communication.

In FL for LiDAR [14], a method is proposed for selecting high-quality clients by sharing evaluations of clients, such as mobile devices, among servers. Through learning while adjusting the proportions of malicious attackers and clients with poor communication environments, the transition of client evaluation scores was measured.

VI. Conclusion

This paper presented an FL platform on clustered many-core processors, enhancing the accuracy and scalability of DL models in autonomous vehicle development. Notably, utilizing clustered many-core processors facilitated efficient parallel processing, thereby reducing power consumption while maintaining high computational performance. Additionally, the employment of FL ensures enhanced data privacy and security, which is in autonomous systems. The platform's scalability suggests its applicability in a broad range of IoT scenarios beyond autonomous vehicles.

In future research, we envision changing the data type to fp16 and utilizing the co-processor in the many-core processor. The co-processor has the potential to be several times faster, further increasing processing speed and reducing power consumption.

Acknowledgment

This work was supported by JST PRESTO Grant Number JPMJPR21P1.

References

[1] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, "Autoware on board: Enabling autonomous vehicles with embedded systems," in *In Proc. of ACM/IEEE International Conference on Cyber-Physical Systems*, 2018, pp. 287–296.
[2] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," 2017, pp. 1273–1282.
[3] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K. H. Li, T. Parcollet, P. P. B. de Gusmao, and N. Lane, "Flower: A friendly federated learning framework," *arXiv:2007.14390*, 2020.
[4] T. Nomi, "tiny-dnn," https://github.com/tiny-dnn/tiny-dnn.
[5] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.
[6] B. D. de Dinechin, "Consolidating High-Integrity, HighPerformance, and Cyber-Security Functions on a Manycore Processor," in *Proc. of Annual Design Automation Conference*, 2019, pp. 1–4.
[7] T. Azumi, Y. Maruyama, and S. Kato, "ROS-lite: ROS framework for NoC-based embedded many-core platform," in *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2020, pp. 4375–4382.
[8] T. Yabe and T. Azumi, "Exploring the Performance of Deep Neural Networks on Embedded Many-Core Processors," in *Proc. of IEEE/ACM International Conference on Cyber-Physical Systems*, 2022, pp. 193–202.
[9] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition," *Neural networks*, vol. 32, pp. 323–332, 2012.
[10] W. A. Haque, S. Arefin, A.S.M.Shihavuddin, and M. A. Hasan, "DeepThin: A novel lightweight CNN architecture for traffic sign recognition without GPU requirements," *Expert Systems with Applications*, vol. 168, p. 114481, 2021.
[11] M. Hasumi and T. Azumi, "Exploring the Training Performance of Deep Neural Networks on Embedded Many-core Processors," in *Proc. of Asia Pacific Conference on Robot IoT System Development and Platform*, vol. 2023, 2023, pp. 14–21.
[12] Tuo Zhang and Chaoyang He and Tianhao Ma and Lei Gao and Mark Ma and Salman Avestimehr, "Federated Learning for Internet of Things," in *Proc. of ACM Conference on Embedded Networked Sensor Systems*, 2021, pp. 413–419.
[13] S. Wang, C. Li, D. W. K. Ng, Y. C. Eldar, H. V. Poor, Q. Hao, and C. Xu, "Federated deep learning meets autonomous vehicle perception: Design and verification," *IEEE Network*, 2022.
[14] C. Shi, C.-M. Own, and R. Zhou, "Federated Grouping Panoptic Segmentation for LiDAR Point Cloud Based on Convolutional Network," in *Proc. of IEEE International Conference on Tools with Artificial Intelligence*, 2022, pp. 6–13.