Structure of Deep Learning Inference Engines for Embedded Systems

Seung-mok Yoo
Embedded System Research Group
ETRI
Daejeon, Korea
yoos@etri.re.kr

Changsik Cho
Embedded System Research Group
ETRI
Daejeon, Korea
cscho@etri.re.kr

Kyung Hee Lee
Embedded System Research Group
ETRI
Daejeon, Korea
kyunghee@etri.re.kr

Jaebok Park, Seok Jin Yoon
Embedded System Research Group
ETRI
Daejeon, Korea
{parkjb, sjyoon }@etri.re.kr

Youngwoon Lee
Dept. of Computer & Electronics Convergence Engineering
SunMoon University
Asan, Korea
yw.lee@ivpl.sookmyung.ac.kr

Byung-Gyu Kim
Dept. of IT Engineering
Sookmyung Women's University
Seoul, Korea
bg.kim@sookmyung.ac.kr

***Abstract*— For the last several years, various types of deep learning applications have been introduced. Most deep learning related research and development have been done on servers or PCs with GPUs. Recently there have been a number of moves to apply those applications to the industrial sector. When deep learning techniques are applied to actual targets, we can face some spatial and environmental constraints unlike the laboratory environment.**

In this paper, we describe requirements when deep learning applications run for embedded systems. We introduce our ongoing project on developing a deep learning framework for embedded systems, especially automotive vehicles. Generally, deep learning application development process can be divided to two steps: training a data model with a big data set and executing the data model with actual data. In our framework, we focus on the execution step. We try to design an inference engine to satisfy the operational requirements for embedded systems. We describe our design direction and the structure. We also show preliminary evaluation result.

Keywords—deep learning neural network, embedded system

I. INTRODUCTION

For the last several years, artificial intelligence has attracted attention in both academic and industrial fields. It is not special to see products using artificial intelligence techniques around us any longer. Deep learning neural networks are at the heart of the phenomenon. Some image recognition algorithms based on the deep learning neural network technique broke many records in competitions for the last few years []. In some image recognition cases, it even exceeds human accuracies. It was not until a few years ago that Go's dominance was hard to overcome. However the stereotype was broken after Se-dol Lee, a professional Go player, was defeated by Google Alpha Go in the historic Go match, 2016. This made people try to solve unsolved problems with deep learning neural networks, and improve the performance of their applications or overcome the hurdles they face in a deep learning point of view.

Many applications based on deep learning techniques have been developed on PCs with GPUs or servers in the laboratory. It is because deep learning applications demand large memory for many data sets and high speed computing power for training the neural network consisting of multiple hidden layers. It is well-known that it takes about a week to build a deep learning data model on a server. Recently, image recognition models have been planted on actual target platforms, which are a special-purpose computers such as mobile IoT devices or autonomous vehicles rather than servers or PCs used in the laboratory. Because embedded systems are generally deployed in the field, there are operational requirements for the environment, physical size, etc. Deep learning applications need computing accelerators such as GPUs due to the use of large amounts of data and complex iterative operations, but the processor performance and architecture used in the embedded environment also affects environmental conditions (e.g., temperature, power consumption, etc.). It is necessary to design a hardware suitable for the purpose without using the performance-based system used in the existing research stage. Performance parts cannot be overlooked because you have to consider the speed when using video to recognize the outside environment while driving like an autonomous vehicle. By default, you should be able to meet the response time of several milliseconds.

In order to meet the above requirements when developing a deep learning program for embedded systems, we try to separate the development of a model based on learning and the process to be performed after installation. During development, the server is used to try to optimize the model, and at run time, the model is used to process surrounding input values, and it tends to be divided into models optimized for inference and used. Currently there are several models such as TensorFlow Lite and Windows ML, but they do not meet all the above restrictions.

We are developing an inference engine for embedded systems, which is designed to meet the above requirements. For interoperability between training data models and executing the data model, we provide an NNEF (Neural Network Exchange Format) interface, which is an open format driven by Khronos group [1]. For portability, our inference engine runs on top of OpenCL which is supported most GPUs in the market. For high performance, we design our engine runs multi GPUs. By our preliminary, the OpenCL version shows better performance than the CPU version, while TensorFlow for OpenCL is worse than that for CPU [2].

The remainder of the paper is organized as follows. In chapter 2, we describe the structure of PlaidML, which is the basis of the model that we are developing. In chapter 3, the requirements of automotive high performance computing platform for embedded systems and we introduce the A.HPC.

Authorized licensed use limited to: Product Marketing IEL. Downloaded on October 30,2020 at 16:10:55 UTC from IEEE Xplore. Restrictions apply.

We show the performance evaluations in chapter 4. We conclude in chapter 4.

II. PLAIDML

PlaidML is an open source-based deep learning framework developed by Intel [3]. PlaidML supports user convenience, various user interface, and H / W platform. The portability is excellent because the main parallel processing is processed in an H / W-independent manner. Currently, it does not support multiple GPUs. PlaidML is composed of the frontend, python bindings, and the core layers as shown in figure 1.

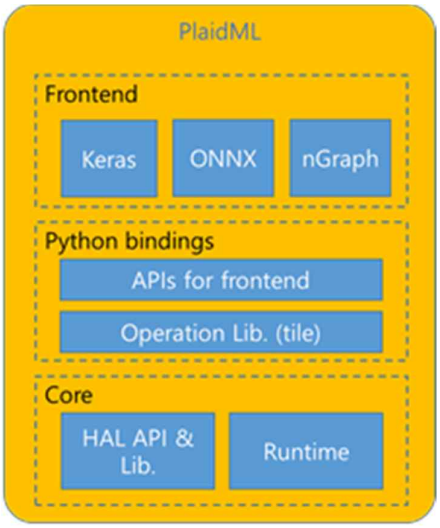

Fig. 1. PlaidML structure

The frontend layer is directly visible to users and developers, and designed to give them programming convenience. It provides Python APIs related to neural network computations. Simple APIs are implemented in Python, but other APIs that demand computing resources such as GPU for complicated matrix operations are included in the python bindings layer. The frontend layer also includes Keras, ONNX (Open Neural Network Exchange), and nGraph, which have been used in many applications recently. Developers can write their programs using either PlaidML APIs or those frontend interfaces already ported.

The python bindings layer is an intermediate layer for connecting the frontend and core layers. This layer includes a set of APIs, an operation library, and a parser for Tile, which is a domain specific language used only in PlaidML to generate GPU kernels. Complicated APIs are implemented in this layer. Most of codes in this layer are written in C++ and linked as a shared object in Linux.

Tile codes are close to mathematical notations. Tile provides basic matrix operations and functions for neural networks. Users can make their Tile functions in the frontend layer. If user-defined functions are passed from the frontend layer to the python bindings, they are compiled to intermediate representation (IR) and transferred down to the core layer.

The Core layer consists of a HAL (Hardware Abstraction Layer) and runtime modules. IR from the above layer is finally compiled to the device-specific code in this layer. Once it is compiled, it is executed in runtime module. Users can also define a HAL for their own H/W platforms. Currently PlaidML provides 4 HALs: CPU, OpenCL, CUDA, and Apple Metal.

III. AUTOMOTIVE HPC ENGINE

We are developing a deep learning inference engine, named Automotive High Performance Computing (A.HPC) engine, for automotive vehicles. We design our engine to meet the constraints for automotive vehicles described as follows.

We assume that training is done on the server side. There are several deep learning frameworks in the market. They have their own advantages and disadvantages. Some of them should show better performance in terms of accuracy, parallelism, training time, and so on. In general, people has believed that when one framework is used for training on the server side, the same framework should be used for inference on the deployed platform. It has been kept for last several years. For interoperability, we try to avoid this restriction by providing a neural network exchange format (NNEF) interface [4, 5]. NNEF is a version of the neural network model presented by the Khronos group and version 1.0 was released in 2018. The standardization work is continuing to add other functions [6].

There is spatial and environmental constraints. Although not explicitly stated for mobile IoT devices or autonomous vehicles, there is a physical size limitation for each design. In addition, there are constraints on temperature conditions and power consumption. This has a direct limitation on the memory and computational performance of the processor within the embedded system. A processor that meets these requirements typically has a lower performance and capacity than a server. On the other hand, target applications with fast running speed such as autonomous vehicles require ms-level latency.

In the embedded system, there is a limitation on the lower platform that can be used depending on the environment. Especially for mobile devices, ARM processors are the mainstream. Therefore, in the case of embedded systems, unlike servers, assuming a tool or platform dependent on a specific H / W, there are many limitations in developing and installing the program. To overcome this, it is necessary to develop based on standardized interface. The platform under development is based on Khronos OpenCL and is available on most commercial GPUs developed to date.

In order to satisfy the above constraints and the performance requirement for quick response, the platform under development satisfies the performance requirement by supporting multi GPUs.

IV. PERMANCE EVALUATIONS

Our project is being development. We show the partial performance of our engine in the chapter. The performance of the inference engine mainly depends on that of CPU, GPU, and memory for the processing. We use a mini-ITX board with an AMD Ryzen 5 2400G processor, an AMD RX550 video card, and 32GB memory instead [7]. The CPU and GPU were announced in 2018. PlaidML is also delicate in some

Authorized licensed use limited to: Product Marketing IEL. Downloaded on October 30,2020 at 16:10:55 UTC from IEEE Xplore. Restrictions apply.

sense. The inference engine runs on top of OpenCL, when GPU operations are required. It is quite tricky to set up the S/W environment. The working environment is as follows. The OS is Ubuntu 18.04.1 LTS and the Linux kernel version is 4.19.03, because the OpenCL driver for the GPU only works on Ubuntu 18.04 and the GPU on the processor can be identified in the Linux kernel version 4.18 and later.

We have made a program that can classify objects from a camera input or a video file as shown in figure 2. For the object classification, VGG16 is used in the program [8]. We run the program in two different conditions, one on CPU-only and the other on GPU, and measure each FPS (frame per second) of the program. The performance improvement of the GPU version is 216% over the CPU-only version. In our previous work, we compared the performance of TensorFlow for OpenCL over TensorFlow for CPU[2]. By our experiment, the execution time of TensorFlow for OpenCL was slower than that of TensorFlow for OpenCL. Although this experiment is preliminary, the result is quite good compared to the TensorFlow case.

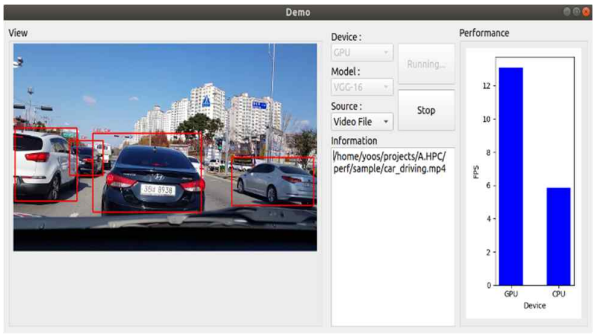

Fig. 2. Screenshot of our test program

V. SUMMARY

In this paper, we have dealt with a deep learning framework for embedded systems which has recently received much attention. Deep learning S/W development process is divided to the training phase to build a data model from a big data set, and the inference phase to execute the data model with actual data.

Our engine has been designed to run deep learning data models in embedded systems efficiently. Thus we focus on the deep learning inference. However, there is no restriction such that users use the same deep learning framework during training data models. By providing an NNEF interface in our framework that we are developing, we allow users to build their data models in any frameworks depending on their preferences. There are spatial and environmental restrictions. Depending on target applications, processors we can pick are different from those for the laboratory environment. Sometimes we should pick processors with which we are not familiar. Still we need H/W acceleration. Thus our engine is designed works on OpenCL which is provided by most GPUs in the market. In other words, if any processors have OpenCL working, our engine can run on them. In addition, our engine is designed to support multiple GPUs to satisfy the performance requirements.

ACKNOWLEDGMENT

This work was supported by Institute for Information & Communications Technology Promotion (IITP) grant funded by the Korea government(MSIP) (No. 2017-0-00068, A Development of Driving Decision Engine for Autonomous Driving (4th) using Driving Experience Information).

REFERENCES

[1] https://www.khronos.org/nnef

[2] Seung-mok Yoo, et al., “Platform-Independent High Performance Inference Engine Running on GPUs,” The 12th IEMEK Symposium on Embedded Technology, 2017.

[3] https://www.intel.ai/plaidml/

[4] https://www.khronos.org/

[5] Seung-mok Yoo, et al., “Frontend Adapter Based on PlaidML,” The 13th IEMEK Symposium on Embedded Technology, 2018.

[6] Kyung Hee Lee et al., “An Implementation of a Parser for Neural Network Exchange Format,” The 13th IEMEK Symposium on Embedded Technology, 2018.

[7] https://www.amd.com/

[8] Karen Simonyan, et al., “Very Deep Convolutional Networks for Large-Scale Image Recognition,” arXiv technical report, 2014.

Authorized licensed use limited to: Product Marketing IEL. Downloaded on October 30,2020 at 16:10:55 UTC from IEEE Xplore. Restrictions apply.