2021 IEEE 11th International Conference on Consumer Electronics (ICCE-Berlin) | 978-1-6654-2831-6/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICCE-Berlin53567.2021.9720006

Interleaved Data Processing Scheme for Optimizing Tensorflow Framework

Jinseo Choi, Minseon Cho, and Donghyun Kang
Changwon National University
Changwon, South Korea
{jinseo,seonjm,donghyun}@changwon.ac.kr

***Abstract*—Nowadays, technologies based on machine learning (ML) are becoming a popular solution for building recommendation services on consumer electronics (CE) devices. Therefore, in this paper, we introduce `i.data` (interleaved data) processing scheme to more efficiently enable ML models on CE devices. The key idea of `i.data` is to interleave the training and inference steps in a parallel way so as to improve the GPU utilization and overall performance. We also compared `i.data` with Numpy and tf.data that are widely used to build ML models in various environments including CE and the internet of things (IoT). Our experimental results clearly show that `i.data` improves the performance compared with the traditional data processing approaches.**

***Index Terms*—machine learning, multiple thread, dataflow, tensorflow, tf.data**

I. Introduction

For several years, machine learning (ML) has opened new challenges and achieved great success in various environments. Especially, researchers and engineers on consumer electronics (CE) and IoT devices have started to enhance their own services by adopting ML-based approaches. For example, Samsung Electronics recently introduce new refrigerators with BESPOKE that provides recommendation services based on ML [1]. Some researchers focused on understanding image data that is the most important part of image classification. They could implement their idea as a prototype simply and faster by using TensorFlow framework.

However, prior efforts have focused only on the ML model for guaranteeing high accuracy, even though the model may wastes GPU resources while running on it. Recent advances in ML have shifted to consider the utilization of hardware resources (*e.g.*, CPU / GPU utilization, memory bandwidth, and I/O costs) [2]–[5]. For example, Zhou *et al.* proposed a lightweight cross-platform system for tiny on-device learning [2]. Some researchers also studied hardware offloading techniques that offer user-level software logic for ML by designing FPGAs [3]–[5]. Unfortunately, previous studies are difficult to be gained wide adoption on CE or IoT environments because of hardware constraints (*e.g.*, power consumption, space, etc.).

In this paper, we propose a new scheme, called `i.data` (interleaved data) processing scheme, that runs the training and inference step in a parallel way. `i.data` is straightforward and compatibility, but it efficiently enhances overall performance with high GPU utilization. Our key insight is that there is a sequence of processing steps (*i.e.*, training and inference) to run the ML model and each step can be pipelined to improve the overall resource utilization. For evaluation, we built CNN model with image classification workload (*i.e.*, MNIST [6]) and compared `i.data` with the original Numpy and tf.data [7]. As we expected, `i.data` clearly presented the enhanced results in terms of overall performance by up to 17% compared with the Numpy.

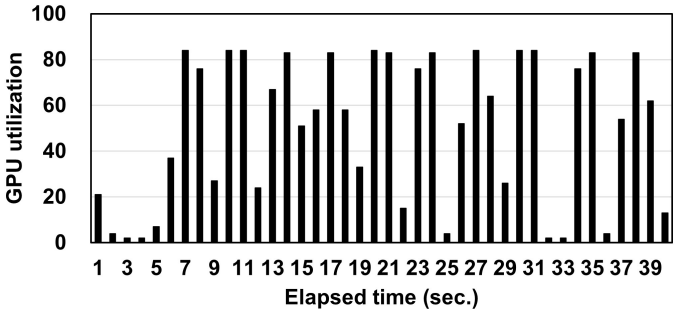


Fig. 1: GPU utilization of Numpy during runs CNN model with MNIST database.

In the rest of this paper, we first describe a brief overview of the data processing flow for ML models to understand `i.data` (section II). Then, we will present `i.data` in detail (section III) and show our evaluation results (section IV) Finally, we conclude this paper (section V).

II. Dataflow and Analysis

In this section, we briefly describe how data is processed for performing training and inference steps on the ML-based models (see Figure 2a). In general, an ML-based model goes over the *training* and *inference* steps within one epoch. The epoch is repeated until getting to the optimal epoch that is pre-defined by a user.

The *training* step can be composed of four phases, and each training step is handled as follows: (1) the first phase is dataload phase where data is loaded as an argument of input through either networks or file systems; (2) after finishing the data load, data is transformed to the units of TensorFlow (*i.e.*, tensor) via Preprocessing phase; (3) the tensors are passed

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. NRF-2019R1G1A1099932). Donghyun Kang is the corresponding author of this paper.

to Shuffle phase where they are shuffled each other to avoid erroneous inference; (4) finally, a set of tensors is delivered as training arguments for ML-based models (*e.g.*, CNN, RNN, and DNN) by going through Batch phase. At the end of the training step, Batch phase is triggered to prepare the *inference* step. After that, the *inference* step is started to make predictions based on the *trained model* that is initialized with the learned parameters through the training step. Note that, the training and inference step are conducted in a serialized fashion because the learned parameters should not be modified during the inference step (*i.e.*, one thread is responsible for two steps). As a result, it may limit the overall hardware resources, especially GPU utilization.

In order to confirm our intuition as mentioned before, we conducted simple tests by implementing CNN model with NumPy [8]. In this evaluation, we followed the default guidelines to build training and inference steps based on the CNN model and used MNIST database as the input [6], [9], [10]. Figure 1 shows GPU utilization on a test machine (we will describe the evaluation setup in detail later). Unfortunately, as shown in the figure, the model utilizes only 53% of GPU resources on average. This observation is very interesting in that the model do not fully utilize GPU resources, even though there is no interference with other tasks. This finding motivates us to further study and a new design for the dataflow on ML-based models.

III. Design of Interleaved Data

As mentioned before, the moderate dataflow for ML-based models is often inefficient and impractical. We believe that the root cause of the collapse comes from each step which is carried out with a single-threaded data processing. To address this problem, we designed `i.data` (interleaved data) processing scheme that performs *training* and *inference* steps in a parallel way. For interleaved data processing, `i.data` has two unique steps compared with the traditional ones:

- `i.data` builds a thread pool in advance to make thread-based training and inference steps; a thread in the pool will be mapped into a training or inference process.
- At the end of the training process, `i.data` copies the model calculated by the last training process so as to make the **locked model**.

To immediately trigger the next training process, `i.data` makes predictions using the locked model, not the trained model. Note that it is possible to interleave data processing, as the learned parameters updated from the next training process never impact the current inference process. Figure 2b shows how thread-based training and inference steps in `i.data` are run and triggered. As shown in Figure 2b, `i.data` handles the first epoch in a sequential way like the traditional approach. However, the thread-based process for the next training step is immediately triggered at the end of copying the locked model without waiting until the end of the inference process. Therefore, `i.data` can handle the training and inference processes on both CPU and GPU, simultaneously. We believe

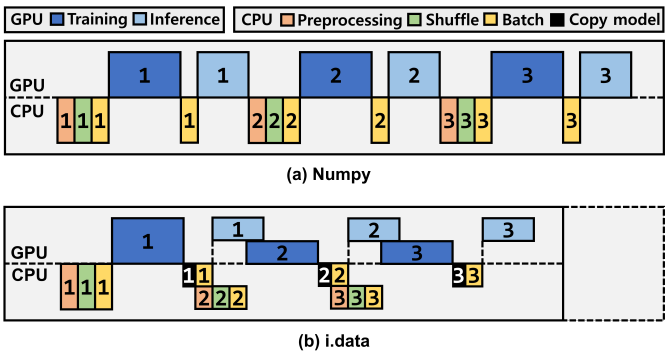


Fig. 2: Data processing flow for ML models. This figure shows (a) the traditional baseline (Numpy) and (b) `i.data`, respectively. The number inside the rectangle means the epoch cycle.

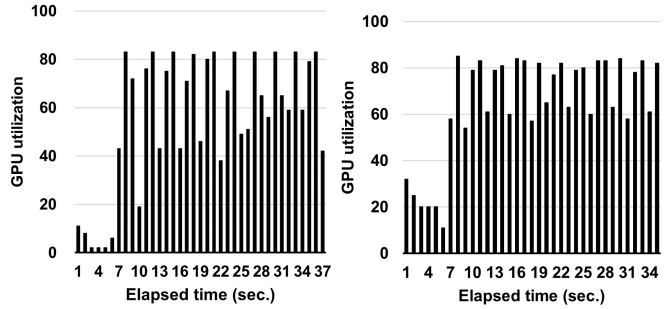
such dataflow of `i.data` can mitigate the wasted GPU utilization in the most of popular ML models.

IV. Evaluation

In this section, we evaluate GPU utilization of `i.data`, and then show the influence of high utilization in terms of the performance.

For evaluation, we performed all experiments on a machine with Intel(R) Core(TM) i5-9500 CPU @3.00GHz, 16 GB memory, Intel NVMe SSD (512 GB), and Nvidia Geforce GTX 1650 GPU with 4 GB memory. We also used Ubuntu 16.04 LTS (Linux kernel 4.15.0) and TensorFlow (version 2.5.0) with Keras (version 2.5.0) to build CNN model. In order to demonstrate the efficiency of `i.data`, we employed MNIST database that is commonly used for ML as inputs of the model [6]. In this paper, we compare `i.data` with two traditional schemes: Numpy and tf.data [7]. NumPy was built by following the implementation guide of Keras, and tf.data and `i.data` were implemented by applying their mechanisms to the Numpy. tf.data is the already well-known scheme for enabling parallelism inside the data processing of the model. Unfortunately, tf.data only offers minimum configuration sets (*i.e.*, buffer_size, seed, num_parallel_cells, and reshuffle_each_iteration, etc.) and it rarely focus on the relationship between the training and inference step. Finally, we repeated all experiments 3 times to clearly confirm the performance gain.

First of all, we focus on the GPU utilization of three data processing approches: Numpy, tf.data, and `i.data`. To collect the GPU utilization, we employed `gpustat` command at a 1 second interval. Figure 3 shows the monitored GPU utilization for tf.data and `i.data`, respectively. As we expected, tf.data shows better GPU utilization compared with Numpy; it is on average 56%. This enhanced result can be described with multiple thread processing as tf.data also supports the parallelism in the batch phase. On the other hand, `i.data` shows the best GPU utilization (on average 63%), especially, the flow is very stable compared to others. The reason behind



(a) tf.data (b) i.data

Fig. 3: GPU utilization of CNN model with MNIST database [6]. This figure presents GPU utilization of tf.data and `i.data`, respectively.

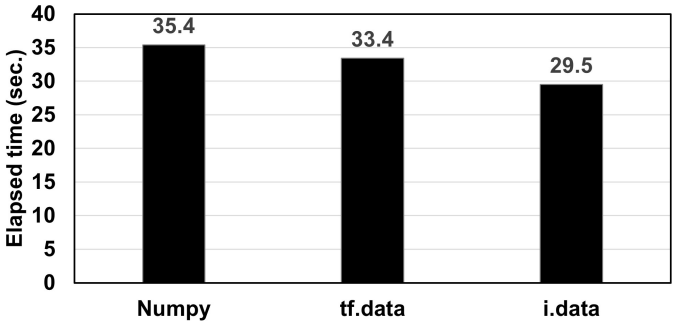


Fig. 4: Performance comparison of `i.data` with two traditional approaches. This figure shows the average elapsed time of three runs.

the positive gain is that `i.data` makes GPU calculate more tensors by interleaving the training and inference process in a parallel way (see Figure 2b).

Now, let us describe the performance to understand how GPU utilization affects the processing time of the CNN model. Figure 4 shows the comparison of elapsed time of `i.data` with Numpy and tf.data. As shown in the figure, `i.data` reduces the elapsed time by up to 17% (on average) compared to Numpy. What makes the result interesting is that the performance gain surely comes from the interleaved steps of `i.data` which increases GPU utilization (see Figure 3b).

V. Conclusion

In this paper, we first introduced the fact that ML-based models unintentionally waste GPU utilization because of the serialization of dataflow. Based on the observation, we proposed `i.data` (interleaved data) processing scheme that aims to achieve high GPU utilization when running ML models. We also implemented `i.data` using Tensorflow framework and compared it with the existing approaches, Numpy and tf.data. Our evaluation presented that `i.data` improves the performance on average 17% compared with Numpy. This improvement is very meaningful in that `i.data` was designed in software without any hardware supports. Therefore, we believe `i.data` is suitable for CE or IoT devices that hard to allow hardware extensions. As future work, we will focus on exploring way to maximize parallelism based on `i.data` because there are more opportunities to improve resource utilization in ML-based models.

References

[1] "Bespoke," https://www.samsung.com/us/bespoke/, 2021, [Online; accessed 7-October-2021].

[2] Q. Zhou, S. Guo, Z. Qu, J. Guo, Z. Xu, J. Zhang, T. Guo, B. Luo, and J. Zhou, "Octo: Int8 training with loss-aware compensation and backward quantization for tiny on-device learning," in *Proceedings of 2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 177–191.

[3] E. Baek, H. Lee, Y. Kim, and J. Kim, "Flexlearn: fast and highly efficient brain simulations using flexible on-chip learning," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 304–318.

[4] H. Jang, J. Kim, J.-E. Jo, J. Lee, and J. Kim, "Mnnfast: A fast and scalable system architecture for memory-augmented neural networks," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, p. 250–263.

[5] J. Liu, H. Zhao, M. A. Ogleari, D. Li, and J. Zhao, "Processing-in-memory for energy-efficient neural network training: A heterogeneous approach," in *Proceedings of 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018, pp. 655–668.

[6] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine*, pp. 141–142, 2012.

[7] D. G. Murray, J. Simsa, A. Klimovic, and I. Indyk, "tf.data: A machine learning data processing framework," *arXiv preprint arXiv:2101.12127*, 2021.

[8] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[9] "Tensorflow," https://www.tensorflow.org/?hl=en, 2020, [Online; accessed 20-August-2021].

[10] F. Chollet *et al.*, "Keras," https://github.com/fchollet/keras, 2015, [Online; accessed 12-August-2021].