Training Resilience with Persistent Memory Pooling using CXL Technology

Miryeong Kwon[1], Junhyeok Jang[1], Hanjin Choi[1], Sangwon Lee[1,2], Myoungsoo Jung[1,2]
Computer Architecture and Memory Systems Laboratory
Korea Advanced Institute of Science and Technology (KAIST)[1], Panmnesia[2]
http://camelab.org

I. Introduction

Deep learning-based recommendation systems take the majority of machine resources in diverse production servers and datacenters [1]. To make the recommendation more accurate, hyperscalers have rapidly scaled up the recommendation models (RMs). As a result, recent RMs often consume more than tens of terabytes of memory space [2], [3]. In addition, it is important for the RMs to be failure tolerant as they should be trained for many days without accuracy degradation [4].

We propose TrainingCXL that can efficiently process large-scale RMs in the pool of disaggregated memory while making training failure-tolerant with low overhead. To this end, i) we integrate persistent memory (PMEM) and GPU into a cache-coherent domain as Type-2 devices of CXL 3.0. Enabling CXL allows PMEM to be directly placed in GPU's memory hierarchy, such that GPU can access PMEM without software intervention. TrainingCXL introduces computing and checkpointing logic near the CXL controller, thereby training data and managing persistency in an active manner. Considering PMEM's vulnerability, ii) we utilize the unique characteristics of RMs and take the checkpointing overhead off the critical path of their training. Lastly, iii) TrainingCXL employs an advanced checkpointing technique that relaxes the updating sequence of embeddings across training batches. The evaluation shows that TrainingCXL achieves 5.2× speedup and 72.6% energy savings compared to the modern PMEM-based recommendation systems.

II. Background

Recommendation Model Training. To achieve high accuracy, RMs exploit both sparse features (categorical information) and dense features (numeric information). To this end, RMs employ embedding operations (table lookup/update) and non-linear transformations (*Bottom-MLP*) for sparse and dense features, respectively. The operations encode the features into a vector and then pass them to a main model called *Top-MLP*.

Figure 1 shows a training process of modern deep learning-based RM. Considering different levels of the computing intensiveness, the bottom-MLP operations are performed in GPU, whereas the embedding operations are processed at the host side (CPU). For these embedding operations, the host reads the target embedding vectors from the underlying storage by referring to the table indices in the sparse features. Then the host aggregates the retrieved embedding vectors using simple arithmetic (e.g., sum/mean) to encode them into an input vector for the top-MLP. As the bottom-MLP and

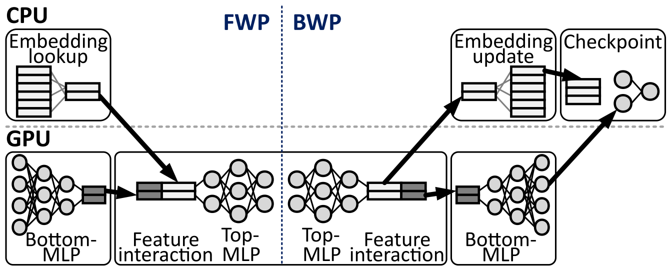


Fig. 1: Training process of DLRM.

embedding operations are processed in different places (i.e., GPU and CPU), each encoded input data for the top-MLP are prepared in parallel to shorten the training time. The backward-propagation step of RM training (BWP) is similar to FWP, but all its operations are processed in the reverse order. It updates the model parameters/embedding by calculating the gradient and then subtracting it from the model parameters/embedding.

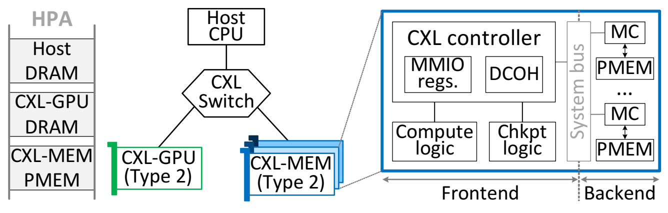
Challenges. Production-scale RMs often cannot be accommodated in a system's local memory since their embedding tables exceed tens of TBs [2], [3]. To this end, several approaches employ high-performance solid-state drives (SSDs) and expand their host memory using SSDs as backend storage media [2], [5]. While this SSD-integrated memory expansion technique can handle large-sized input data, they, unfortunately, suffer from severe performance degradation. This is because RM's embedding lookup frequently generates small-sized random read, whereas SSDs are optimized for bulk sequential I/Os.

Furthermore, prior approaches require explicit checkpoints for fault recovery. Since embedding updates on SSDs can degrade the training performance significantly, the existing RMs utilize the SSDs for only memory expansion purposes. Note that the write latency of SSDs is longer than the latency of all conventional memory operations by orders of magnitude.

III. TrainingCXL

We propose TrainingCXL that can efficiently process large-scale RMs in the underlying memory pool, disaggregated over compute express link (CXL) [6]. Our CXL-enabled memory expander employs PMEMs, which exhibit similar performance to DRAM and provide large memory capacity. In addition, TrainingCXL leverages PMEMs' non-volatility to support failure-tolerant training with low overhead. Our contributions can be summarized as follows.

Intelligent CXL memory expansion. Figure 2a shows an overview of the proposed TrainingCXL's system archi-



(a) Cache-coherent domain. (b) Internals of CXL-MEM.

Fig. 2: TRAININGCXL's system architecture.

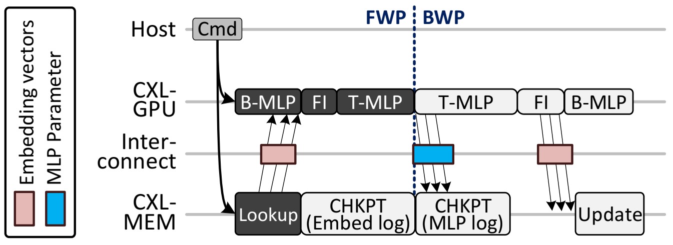


Fig. 3: Execution of batch-aware checkpoint.

tecture. TRAININGCXL offers two CXL devices: a CXL-enabled GPU (*CXL-GPU*) and a PMEM-based memory expander (*CXL-MEM*). The devices are connected to a host CPU through CXL Switch(es). The devices are designed as Type-2 of CXL 3.0 by introducing *device coherent agent* (DCOH) in their hardware. Because of their device type, CXL-MEM's internal memory can be exposed to CXL-GPU's local memory and vice versa. This makes the embeddings and gradients exchanged between those two devices without any software intervention running on the host CPU. To accelerate RM training, CXL-MEM employs simple computing and checkpointing logic along with a Type-2 endpoint controller, which can perform embedding operations and failure tolerance management near PMEM (Figure 2b).

Batch-aware checkpoint. To achieve high throughput, TRAININGCXL lets CXL-GPU and CXL-MEM perform MLPs and embedding operations in parallel. However, persisting the model and embedding updates at the end of processing each batch make the training latency yet longer. To address this, we propose *batch-aware checkpoint* that is aware of embeddings involved in the individual batch and performs undo logging in the background. In typical applications, such a background undo logging scheme is infeasible as the target location where the system needs to update is unavailable before their computation completes. However, in the case of RM training, the embedding vector indices to be updated can be known in advance from the sparse features at the beginning of each batch. Our scheme logs the embeddings vectors corresponding to the indices to CXL-MEM by utilizing the idle time of CXL-MEM as shown in Figure 3.

Embedding lookup and checkpoint relaxation. While our batch-aware checkpoint hides the relatively long latency of CXL-MEM's writes, the training performance is yet limited owing to PMEM's architectural constraints; PMEM's read stalls if the read is requested right after a write for the same physical layout of PMEM. This phenomenon is known as read-after-write (RAW) [7], and is frequently observed between the embedding update of the Nth batch and the embedding lookup of the $(N+1)$th batch [8]. However, since the embedding lookup and update are composed of vector addition/subtraction, the execution order between embedding lookup and update can be relaxed by utilizing their commutative property. Thus, TRAININGCXL proposes a relaxed embedding lookup. It removes the RAW issue by performing embedding lookup (aggregation) for the $(N+1)$th batch before the embedding update of the Nth batch. The aggregated vector is then updated by using the gradient aggregated in the same way how vectors are aggregated during embedding lookup.

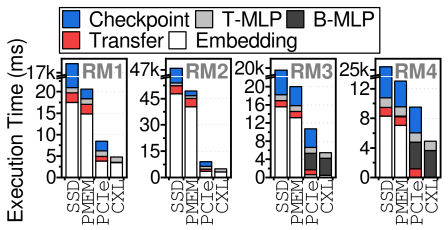


Fig. 4: Training time breakdown.

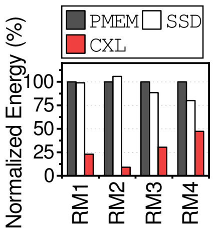


Fig. 5: Energy.

IV. EVALUATION

We evaluate TrainingCXL using a prototype implemented atop of multiple FPGAs. We prepare three different configurations, SSD (`SSD`), PMEM (`PMEM`), and PCIe-attached PMEM (`PCIe`), based on the underlying media where the embedding tables are stored into. While embedding operations of `SSD` and `PMEM` are performed on the host CPU, `PCIe` is capable of near-data processing like our CXL-MEM. We use open source DLRM benchmark [9] and prepare four recommendation system models (RM) for the evaluation.

Figure 4 shows RM's execution time training a single batch. As shown in the figure, `PMEM` exhibits 1063.8× speedup than `SSD`, on average. This is because PMEM can be accessed in a byte-addressable manner, and its read/write is faster than SSD. `PCIe` accelerates the embedding operations, thereby shortening the execution time by 2.7× than `PMEM`. Further, `CXL` achieves a speedup of 1.9×, on average, compared to PCIe. This is because `CXL` takes such overhead off the critical path by CXL-based automatic data movement and batch-aware checkpointing. Note that `CXL` also eliminates the RAW issue thereby reducing the embedding lookup time.

We also analyze energy consumption between `SSD`, `PMEM`, and CXL; the energy values of `SSD` and `CXL` are normalized the those of `PMEM`. As we can see from Figure 5, `CXL` exhibits 3.6× and 3.4× lower energy consumption than `SSD` and `PMEM`, respectively, on average.

ORIGINAL PUBLICATION

Miryeong Kwon, Junhyeok Jang, Hanjin Choi, Sangwon Lee, and Myoungsoo Jung. "Failure Tolerant Training with Persistent Memory Disaggregation over CXL." IEEE Micro (2023). https://ieeexplore.ieee.org/document/10018437

ACKNOWLEDGEMENT

We thank for all the technical support of Panmnesia. This work is protected by one or more patents. Myoungsoo Jung is the corresponding author (mj@camelab.org).

References

[1] Kim Hazelwood, Sarah Bird, David Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy, Bill Jia, Yangqing Jia, Aditya Kalro, et al. Applied machine learning at facebook: A datacenter infrastructure perspective. In *2018 IEEE International Symposium on High Performance Computer Architecture*. IEEE, 2018.

[2] Weijie Zhao, Jingyuan Zhang, Deping Xie, Yulei Qian, Ronglai Jia, and Ping Li. Aibox: Ctr prediction model training on a single node. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2019.

[3] Dheevatsa Mudigere, Yuchen Hao, Jianyu Huang, Zhihao Jia, Andrew Tulloch, Srinivas Sridharan, Xing Liu, Mustafa Ozdal, Jade Nie, Jongsoo Park, et al. Software-hardware co-design for fast and scalable training of deep learning recommendation models. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022.

[4] Assaf Eisenman, Kiran Kumar Matam, Steven Ingram, Dheevatsa Mudigere, Raghuraman Krishnamoorthi, Krishnakumar Nair, Misha Smelyanskiy, and Murali Annavaram. Check-N-Run: a checkpointing system for training deep learning recommendation models. In *19th USENIX Symposium on Networked Systems Design and Implementation*, 2022.

[5] Zehuan Wang, Yingcan Wei, Minseok Lee, Matthias Langer, Fan Yu, Jie Liu, Shijie Liu, Daniel G Abel, Xu Guo, Jianbing Dong, et al. Merlin hugectr: Gpu-accelerated recommender system training and inference. In *Proceedings of the 16th ACM Conference on Recommender Systems*, 2022.

[6] CXL Consortium. Compute express link specification 3.0.

[7] Gyuyoung Park, Miryeong Kwon, Pratyush Mahapatra, Michael Swift, and Myoungsoo Jung. Bibim: A prototype multi-partition aware heterogeneous new memory. In *10th USENIX Workshop on Hot Topics in Storage and File Systems*, 2018.

[8] Youngeun Kwon and Minsoo Rhu. Training personalized recommendation systems from (gpu) scratch: Look forward not backwards. In *Proceedings of the 49th Annual International Symposium on Computer Architecture, 2022*.

[9] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G. Azzolini, Dmytro Dzhulgakov, Andrey Mallevich, Ilia Cherniavskii, Yinghai Lu, Raghuraman Krishnamoorthi, Ansha Yu, Volodymyr Kondratenko, Stephanie Pereira, Xianjie Chen, Wenlin Chen, Vijay Rao, Bill Jia, Liang Xiong, and Misha Smelyanskiy. Deep learning recommendation model for personalization and recommendation systems. *CoRR*, abs/1906.00091, 2019.