MERCI: Efficient Embedding Reduction on Commodity Hardware via Sub-Query Memoization

Yejin Lee, Seong Hoon Seo, Hyunji Choi, Hyoung Uk Sul, Soosung Kim, Jae W. Lee, Tae Jun Ham

Seoul National University

{yejinlee, andyseo247, hyunjichoi, stuartsul, soosungkim, jaewlee, taejunham}@snu.ac.kr

1. Motivation

Recommender systems exploit both dense features and categorical (sparse) features to provide personalized content to individual users. Here, to represent a categorical feature, a representation named *embedding* is often utilized. Here, *embedding* is a relatively low-dimensional, dense vector representing the semantics of a categorical data. For example, in an online shopping website, each product on sale can be treated as a feature and be represented with a distinct embedding vector.

In practice, it is common to represent a set of embedding vectors as a single, pooled embedding vector. For instance, recommender systems often represent a user's recently browsed products using this single embedding vector. To obtain this vector, recommender systems gather embedding vectors for all of the user's recently browsed products and then perform a reduction operation (e.g., sum, average, max, inner product). This operation is called *embedding reduction*. Embedding reduction is a widely employed procedure in machine learning (ML) models, and all popular NN frameworks such as Tensorflow [3], PyTorch [8], and Caffe2 [2] support this operation.

Embedding reduction operation is known to be memory-bound. This operation only requires a very simple computation (e.g., sum reduction) for each embedding vector that it fetches, and thus its runtime often becomes bounded by the system's memory bandwidth rather than the system's computation capability. Recent proposals [1, 4, 5, 6] attempt to optimize this operation, but they require hardware modifications which prevents them from being deployed on commodity hardware.

Therefore, this paper introduces a software-only solution for accelerating embedding reduction through *memoization*. Memoization [7] is a classic technique that stores the computation results for specific inputs in advance and then reuse them when the same input arrives again. Specifically, when applied for the embedding reduction, it can potentially reduce the number of memory accesses as well as computation time at the expense of additional capacity cost. This seemingly simple technique comes with several challenges. Memoization table must be able to provide a broad coverage while being compact enough to fit in memory. Also, it is critical not to incur any extra memory accesses to retrieve the memoized values.

Thus, we propose MERCI, **m**emoization for **e**mbedding **r**eduction with **c**luste**r**i**n**g, a novel memoization framework for efficient embedding reduction on the commodity hardware. At the heart of MERCI is Correlation-Aware Variable-Length Clustering, which effectively identifies variable-length clusters of frequently co-appearing features—which we indeed observed in real-world datasets—for fine-grained (sub-query) memoization. Furthermore, we introduce feature remapping and memoization table construction that enables the processor to efficiently retrieve memoized values.

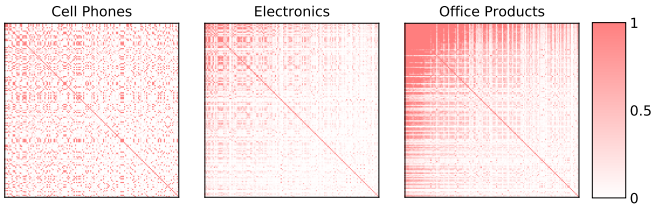

Figure 1: Correlation heat map for product pairs of top 150 items in the Amazon Review dataset.

2. Key Insights

Opportunities for sub-query memoization. We identify new opportunities for *fine-grained* (i.e., sub-query granularity) partial reduction memoization by exploiting the correlation structure in real-world categorical features. For instance, Figure 1 depicts the pair-wise co-appearing frequency of the top 150 items in the Amazon Review dataset; a darker dot means that the pair tends to co-appear more frequently as features of an embedding reduction operation. The figure only shows the pair-wise co-appearance, but there exist many *variable-sized* set of features that tends to co-appear in a single embedding reduction operation. Our work exploits such patterns and memoizes the partial reduction results for the features that tend to co-appear more frequently. By doing so, MERCI replaces multiple memory accesses and some reduction computations with single memory access. Such partial reduction is possible as a reduction operator (e.g., sum) is commutative and associative.

No one-size-fits-all in clustering. To identify the aforementioned clusters of frequently co-appearing features, we need to cluster features based on their co-appearance patterns. Indeed, there already exist algorithms such as hypergraph partitioning algorithms whose goal is to generate a specified number of *equal-sized* partitions such that features in the same partition are likely to appear together. However, such algorithms cannot generate variable-sized partitions, which is essential to model real data inputs exhibiting variable-sized partition structures. Furthermore, they do not allow users to limit the memory capacity cost of memoization in a fine-grained manner. Thus, we propose Correlation-Aware Variable-Length Clustering, which can reflect diversity in the size of correlated features by

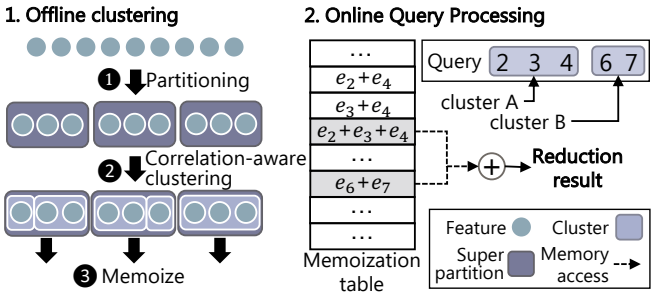

Figure 2: Overview of MERCI

evaluating *benefit* and *cost*, and generating clusters of size one to twenty or more. It also sets memory limits at a granularity of less than 1% of the original embedding table size.

Efficient online query processing. One challenge in memoization is the fast retrieval of the memoized values. Specifically, if a memoization table lookup requires more memory accesses than accessing all the embeddings that a particular memoized value covers, memoization will not be beneficial. To avoid such additional memory accesses, we remap feature IDs, enabling quick identification of the cluster for each feature. We also carefully construct a memoization table so that identifying the location of the memoized value is possible with the minimal number of additional memory accesses. Finally, we devise an efficient memoized value retrieval mechanism based on those two strategies.

3. Main Artifacts

MERCI consists of two phases: offline clustering (details in Section 4 of the full paper) and online query processing (details in Section 5). Figure 2 shows an overview of MERCI.

Offline clustering. ❶ MERCI first partitions N features into a set of coarse-grained, fixed-length partitions called *super-partitions* by utilizing an existing hypergraph partitioning algorithm on the training dataset. Each super-partition contains features that are likely to appear together based on the history of queries. Then, ❷ MERCI applies Correlation-Aware Variable-Length Clustering to each super-partition, dividing features in a super-partition into fine-grained, variable-length clusters. ❸ MERCI creates a memoization table holding all possible partial sum combinations within each cluster.

Online query processing. MERCI utilizes the memoization table created from the previous phase to serve incoming queries. Once a query arrives (e.g., $\{2,3,4,6,7\}$) MERCI first identifies which features belong to the same cluster. In this example, features $\{2,3,4\}$ belong to cluster A and $\{6,7\}$ to cluster B. Hence, two memoized partial sums (embeddings) are retrieved (i.e., $e_2+e_3+e_4$ and e_6+e_7) and summed up to generate the final output. In this scenario, the original embedding reduction operation requires loading five embeddings (i.e., e_2, e_3, e_4, e_6, e_7). On the other hand, MERCI only needs to load two. Since the embedding reduction operation is often memory bandwidth-bound, such reduction in memory accesses directly translates to performance improvements.

4. Key Results

Throughput. MERCI achieves significant throughput improvement over baseline with no memoization. Specifically, across all datasets, MERCI manifests throughput improvement in the range of 47-219%, and achieved an average speedup of 67% and 93% at the expense of 1× and 8× capacity overhead, respectively. Figure 11 of the full paper shows the throughput improvement with MERCI.

Energy Saving. The improvements in throughput resulting from the reduced number of memory accesses lead to the energy consumption reduction. Evaluation shows that MERCI reduces the energy consumption (i.e., CPU package + DRAM) by 25-35%. Figure 13 of the full paper shows the energy savings with MERCI.

5. Improvements over State-of-the-Arts

MERCI is the first attempt to apply memoization for optimizing embedding reduction; thus, there is no direct state-of-the-art work to compare. Still, there are works that aim to accelerate embedding reduction with various approaches.

Hardware solutions for embedding reduction. For example, RecNMP [5] and TensorDIMM [6] adopt the near-memory processing (NMP) paradigm to exploit the abundant internal bandwidth to perform reduction and only passes the reduction outcome to the external device through links with lower bandwidth. Centaur [4] is a chiplet-based hybrid accelerator that also includes embedding reduction as its target. However, solutions that require hardware support are often expensive. In contrast, MERCI is immediately deployable to provide a substantial speedup on the commodity hardware at the cost of extra memory capacity.

Feature-aware optimizations. Bandana [1] utilizes hypergraph partitioning algorithms to place embedding vectors that are likely to be accessed together in the same 4KB NVM block. Doing so enables Bandana to efficiently utilize its limited DRAM cache capacity. However, it still incurs the same number of DRAM memory accesses to perform an embedding reduction operation.

6. Why ASPLOS?

Our work aligns well with the spirit of ASPLOS, which highly encourages interdisciplinary research that solves emerging real-world problems. MERCI identifies opportunities for sub-query memoization from application-level usage patterns (i.e., co-appearing features in real-world datasets), but the solution involves low-level code optimization to make memoization performant. In particular, MERCI i) tackles the embedding reduction, which has become a major primitive in data center workloads, ii) presents a memoization framework with a clustering scheme that considers both dataset correlation and memory constraints, iii) achieves substantial throughput improvement and energy savings.

References

[1] Assaf Eisenman, Maxim Naumov, Darryl Gardner, Misha Smelyanskiy, Sergey Pupyrev, Kim M. Hazelwood, Asaf Cidon, and Sachin Katti. Bandana: Using non-volatile memory for storing deep learning models. In *Proceedings of Machine Learning and Systems (MLSys)*, 2019.

[2] Facebook. Caffe2. https://caffe2.ai, 2016.

[3] Google. TensorFlow: Large-scale machine learning on heterogeneous systems. http://tensorflow.org/, 2015.

[4] Ranggi Hwang, Taehun Kim, Youngeun Kwon, and Minsoo Rhu. Centaur: A chiplet-based, hybrid sparse-dense accelerator for personalized recommendations. In *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020.

[5] Liu Ke, Udit Gupta, Benjamin Youngjae Cho, David Brooks, Vikas Chandra, Utku Diril, Amin Firoozshahian, Kim M. Hazelwood, Bill Jia, Hsien-Hsin S. Lee, Meng Li, Bert Maher, Dheevatsa Mudigere, Maxim Naumov, Martin Schatz, Mikhail Smelyanskiy, Xiaodong Wang, Brandon Reagen, Carole-Jean Wu, Mark Hempstead, and Xuan Zhang. RecNMP: Accelerating personalized recommendation with near-memory processing. In *Proceedings of ACM/IEEE Annual International Symposium on Computer Architecture, (ISCA)*, 2020.

[6] Youngeun Kwon, Yunjae Lee, and Minsoo Rhu. TensorDIMM: A practical near-memory processing architecture for embeddings and tensor operations in deep learning. In *Proceedings of IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2019.

[7] Donald Michie. "Memo" functions and machine learning. *Nature*, 218(5136):19–22, 1968.

[8] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.